AFIT/GOR/ENS/98M-21

EMBEDDING A REACTIVE TABU SEARCH HEURISTIC IN UNMANNED AERIAL VEHICLE SIMULATIONS

THESIS

Joel L. Ryan, Captain, USAF

AFIT/GOR/ENS/98M

AFIT/GOR/ENS/98M-20

THESIS APPROVAL

NAME: Joel L. Ryan, Captain, USAF **CLASS**: GOR-98M

THESIS TITLE: Embedding a Reactive Tabu Search Heuristic in Unmanned Aerial Vehicle Simulations

DEFENSE DATE: 3 March 1998

COMMITTEE:	**NAME/TITLE/DEPARTMENT**	**SIGNATURE**
Advisor	Glenn Bailey, Lieutenant Colonel, USAF Assistant Professor of Operations Research Department of Operational Sciences Air Force Institute of Technology	
Reader	James T. Moore, Lieutenant Colonel, USAF Associate Professor of Operations Research Department of Operational Sciences Air Force Institute of Technology	James T. Moore
Reader	William B. Carlton, Lieutenant Colonel, USA Adjunct Assistant Professor of Operations Research Department of Operational Sciences Air Force Institute of Technology	

EMBEDDING A REACTIVE TABU SEARCH HEURISTIC
IN UNMANNED AERIAL VEHICLE SIMULATIONS

THESIS

Presented to the Faculty of the Graduate School of Engineering
Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Operations Research

Joel L. Ryan, B. S.
Captain, USAF

February 1998

Approved for public release; distribution unlimited

Acknowledgments

This section would be better titled "Thanksgivings." Thanksgiving implies the more heartfelt thanks Americans give to God and fellow man on their annual holiday, and it is this level of gratitude I wish to express to God and the individuals listed here. I have risen far above my expectations. It is a fulfillment of prayer and a gift from family and colleagues.

Foremost, I thank my wife, Tracy, who sacrificed so much in the past year to give me the freedom to pour myself into this work. In addition to raising two sons, you had a third without breaking stride. The family graduates from AFIT stronger than when it arrived, and we owe it all to you.

Lt. Col. Glenn Bailey gave me the encouragement and direction I needed. Without his support, I think I would have succumb to the doubt that too often drowns great journeys. His vision for this project was a powerful motivation that pushed me onward.

LTC William Carlton was a selfless aide in this project. Without his code and instruction, this thesis would not have been possible. I am glad we found more ways to give his powerful work life.

Lt. Col. Miller took the time to get me over more than one hump in using MODSIM. Lt. Col. Moore was a patient and constructive reader. All I can offer in return is thanks.

I also thank the guys in 133b, especially "Useful" Joel Coons. You made the work environment bearable and fun.

Table of Contents

List of Figures

Figure | Page

List of Tables

Table Page

Abstract

We apply a Reactive Tabu Search (RTS) heuristic within a discrete-event simulation to solve routing problems for Unmanned Aerial Vehicles (UAVs). Our formulation represents this problem as a multiple Traveling Salesman Problem with time windows (mTSPTW), with the objective of attaining a specified level of target coverage using a minimum number of vehicles. Incorporating weather and probability of UAV survival at each target as random inputs, the RTS heuristic in the simulation searches for the best solution in each realization of the problem scenario in order to identify those routes that are robust to variations in weather, threat, or target service times.

Generalizing this approach as Embedded Optimization (EO), we define EO as a characteristic of a simulation model that contains optimization or heuristic procedures that can affect the state of the system. The RTS algorithm in the UAV simulation demonstrates the utility of EO by determining the necessary fleet size for an operationally representative scenario. From our observation of robust routes, we suggest a methodology for using robust tours as initial solutions in subsequent replications. We present an object-oriented implementation of this approach using MODSIM III, and show how mapping object inheritance to the GVRP hierarchy allows for minimal adjustments from previously written objects when creating new types. Finally, we use EO to conduct an analysis of fleet size requirements within an operationally representative scenario.

Chapter 1

As begun by Sisson (1997), this thesis further expands the reactive tabu search (RTS) of Carlton (1995) into Unmanned Aerial Vehicle (UAV) applications of vehicle routing. Carlton's RTS is translated into MODSIM (CACI 1996) to take advantage of its object-oriented qualities. Simulation is a MODSIM strength that facilitates our implementation of embedded optimization. Embedded optimization is formally defined and employed as a method for incorporating the stochastic nature of inputs to the UAV scenarios. Although the thesis has no immediate sponsor, the work is tailored for the U. S. UAV Air Force Battlelab. The work described here is targeted towards their mission of demonstrating the military worth of innovative concepts.

Capitalizing on the advantages of object-oriented programming languages, a number of libraries are created in MODSIM that can be used to quickly develop tabu searches tailored to any specific member of the general vehicle routing problem (GVRP) family. While MODSIM is not a common programming language, the code is easily read by any programmer of moderate experience in other languages. As an aid to the use of these libraries, Carlton's hierarchical taxonomy of the GVRP is a road map of the steps necessary to transform a search tailored for one class of problems to another. We present an efficient format for the direct comparison of Carlton's pseudocode and accompanying MODSIM library to the UAV pseudocode and libraries.

Chapter 2 is written in an article format meant for journal submission. After a brief explanation of the thesis motivation in Section 2.1, Section 2.2 presents the UAV problem formulation and Section 2.3 reviews work relevant to the study of the GVRP. Section 2.4 reviews the power of RTS for vehicle routing problems and the validation of our MODSIM objects. Section 2.5 discusses our implementation of the RTS within the UAV environment. In 2.5.1, the advantages of an object-oriented execution of the pseudocode are presented, while

2.5.2 presents an original embedded optimization approach. Section 2.6.1 provides the results of our search for a robust tour structure and section 2.6.2 contains the fleet size analysis within an operationally representative scenario. Section 2.7 concludes our research with recommendations for further study. The appendices contain the original MODSIM code developed during the research and an extended literature review.

Chapter 2

2.1. Introduction

Assad (1988) claims vehicle routing as one of the great success stories of operations research (OR), referring to the particular success this area has enjoyed from the implementation of academic advances. Although Hall and Partyka (1997) indicate the advances have not slowed, much work remains in military applications (Sisson 1997). Glover and Laguna (1997) further emphasize the reactive tabu search (RTS) as a neglected area within the broader study of tabu search (TS), one of the more recent and most effective techniques applied to the vehicle routing problem (VRP) (Laporte 1992, Rego 1996).

This research continues investigations begun by Carlton (1995) and Sisson (1997) into the effectiveness of RTS on a close relative of the VRP, the multiple traveling salesman problem with time window constraints (mTSPTW). Specifically, unmanned aerial vehicle (UAV) applications test our implementations of RTS. The advantages of object-oriented programming are added to these earlier works, forming a structure for extensive exploration of problems within the general vehicle routing problem (GVRP) family.

As our main contribution to GVRP research, we identify routes that are persistent throughout a simulation's state space. We propose a formal definition and implementation of embedded optimization as the mechanism that expands the use of optimization methods to routing problems such as the UAV application. UAV problems differ from those traditionally found in the GVRP literature because they include stochastic inputs; for example, random winds (magnitude and direction) and service times are fundamental to all our scenarios. Although these

scenarios are tailored to a UAV environment, the embedded optimization approach we implement applies to any stochastic VRP.

2.2. The UAV Problem Formulation

Our fundamental idea is to apply Carlton's RTS (1995) within a Monte-Carlo simulation of notional scenarios in order to identify routes robust, or persistent, to parameter variation. The first set of scenarios is considered operationally representative by the U.S. Air Force UAV Battlelab (Bergdahl 1997) and contains stochastic wind speed and direction and service times (a service time represents the time a UAV must loiter over a target). The objective function seeks a hierarchical objective of the minimum number of vehicles needed to achieve a minimum tour completion time. Sisson (1997) provides a second set of scenarios as a futuristic look at the routing of UAVs through a threat environment. The objective function for this second set of scenarios seeks the maximum expected target coverage. If alternate optima exist that differ in the number of vehicles required, the solution using the lesser amount is chosen.

At the core of our UAV problem formulation is a mTSPTW. Carlton's (1995) RTS seeks "near optimal" solutions to a mTSPTW with nc customers, indexed by i or j, each requiring a service time s_i at location i. The starting depot is designated 0; the terminal depot by nc (see Lenstra 1985). With nv vehicles available, if no feasible solutions are found after a reasonable search we increase nv and restart the search.

The time window for each customer i's pick up is (e_i, l_i), where e_i is the earliest possible arrival and l_i is the latest. The early arrival time is treated as a "soft" constraint in that vehicles arriving before e_i may wait until e_i is reached. W_i is the wait time at customer i. The parameter $t_{i,j}$ is the travel time from customer i to customer j. The binary decision variable $X^v_{i,j}$ equals 1 if

vehicle v travels on the arc between customers i and j; otherwise it is 0. Tour schedule variables A_i and T_i indicate the time a vehicle arrives at customer i and the time service starts at customer i, respectively. The time windows, times between nodes, and service times are constrained to be integer. The formulation (Carlton 1995, Sisson 1997) seeks a minimum travel time of the tour:

$$\text{MIN } Z_t = \sum_{i=1}^{nc} \sum_{j=1}^{nc} \sum_{v=1}^{nv} X^v{}_{i,j} \cdot t_{i,j}$$

Subject To:

$$\sum_{i=0}^{nc}\sum_{v=1}^{nv} X^v{}_{i,j} = 1 \qquad \forall\, j = 1..nc \qquad \{\text{One vehicle entering per customer}\}$$

$$\sum_{j=1}^{nc}\sum_{v=1}^{nv} X^v{}_{i,j} = 1 \qquad \forall\, i = 0..nc \qquad \{\text{One vehicle leaving per customer}\}$$

$$\sum_{i=0}^{nc} X^v{}_{i,i} = 0 \qquad \forall\, v = 1..nv \qquad \{\text{Cycling prevented}\}$$

{Same vehicle entering a node must exit; routes cannot terminate at target nodes}

$$\sum_{i=0}^{nc} X^v{}_{i,j} = \sum_{\substack{k=0 \\ k \neq i}}^{nc} X^v{}_{j,k} \qquad \forall\, j = 1..nc,\ \forall\, v = 1..nv$$

$$X^v{}_{i,j} = 1 \Rightarrow T_i + s_i + t_{i,j} = T_j \qquad \{\text{Time precedence}\}$$

$$e_i \le T_i \le l_i \qquad \forall\, i = 1..nc \qquad \{\text{Time windows}\}$$

$$W_i = T_i - A_i \qquad \forall\, i = 1..nc \qquad \{\text{Waiting times}\}$$

The subtour breaking constraints are not shown, but are included in our model.

The UAV problem further adds vehicle-related route length constraints and alters the objective function. Given a the maximum time a vehicle can be used, T^v, the route length constraints are defined by

$$\sum_{i=1}^{nc}\sum_{j=1}^{nc} X^{v}{}_{i,j}\cdot s_i+\sum_{i=1}^{nc}\sum_{j=1}^{nc} X^{v}{}_{i,j}\cdot W_i+\sum_{i=0}^{nc}\sum_{j=0}^{nc} X^{v}{}_{i,j}\cdot t_{i,j}\le T^{v} \qquad \forall\ v=1..nv$$

When incorporating the probability of survival for each target, one proposed UAV objective function seeks to maximize expected target coverage (Sisson 1997). Coverage is defined as the number of targets that will be visited; therefore, the expected coverage of any single target equals the probability of surviving that target. Notationally, for target node n_i^v, the i^{th} target node visited in the route of vehicle *v*, the expected coverage is given by

$$\prod_{i=a^v}^{n_i^v} Ps(i)$$

where a^v is the starting node of vehicle *v*'s tour, and *Ps(i)* is the probability of survival at target node *i*. For instance, assuming a UAV travels from target 1 to 2 to 3, and *Ps*(1) = 0.9, *Ps*(2) = 0.8, and *Ps*(3) = 0.7. Target 1's coverage is 0.9, Target 2's is 0.9*0.8=0.72, and Target 3's is 0.90*0.80*0.70=0.50.

The expected coverage of the route of vehicle *v* is given by the sum of the coverages of the nodes along the route, or

$$\sum_{n_i^v=a^v}^{b^v}\ \prod_{i=a^v}^{n_i^v} Ps(i)$$

where b^v is the ending node of vehicle v's tour and $a^v \le n_i^v \le b^v$. Thus, for the three node example above, the expected coverage is 0.90 + 0.72 + 0.50 = 2.12.

The UAV scenario also complicates the calculation of distances and time between points. Given target locations expressed in latitude and longitude, the angular difference (*D*) in radians between locations x (x_{lat}, x_{long}) and y (y_{lat}, y_{long}) is estimated using the relationship

$$D = \cos^{-1}[\sin(x_{lat}) \cdot \sin(y_{lat}) + \cos(x_{lat}) \cdot \cos(y_{lat}) \cdot \cos(abs(y_{long} - x_{long}))]$$

and converted into nautical miles by equating one radian to 57.2958 degrees and one degree to 60 nautical miles (Lindholm 1982). To account for the influence of a wind vector, the Law of Cosines must be used in the travel time calculations and requires we know ϕ, the angle between the heading (θ_{xy}) and the direction from which the wind originates (θ_w) where

$$\phi = \theta_w - \theta_{xy}.$$

Traveling from x to y, the heading is given by

$$\theta_{xy} = \sin^{-1}\left(\frac{y_{lat} - x_{lat}}{D}\right) \qquad (y_{long} - x_{long} \geq 0)$$

$$\theta_{xy} = 180^\circ - \sin^{-1}\left(\frac{y_{lat} - x_{lat}}{D}\right) \qquad (y_{long} - x_{long} < 0).$$

Since UAV operators receive the wind's heading as a compass direction, we convert the wind's given compass heading θ^c_w to the Cartesian coordinate system of θ_{xy} using

$$\theta_w = (360 - \theta^c_w) + 90^\circ.$$

The ground speed GS is estimated as

$$GS^2 = AS^2 + WS^2 - 2 \cdot AS \cdot WS \cdot \cos(\phi)$$

where AS is the UAV airspeed and WS is the magnitude of the wind vector or wind speed. Finally, the division of D by GS, where D and GS are specified in the same units of distance, yields the time to travel from x to y (Klaf 1946).

2.3. The General Vehicle Routing Problem

Carlton's (1995) survey of proposed classification schemes of problems within the GVRP class leads him to conclude no prior system exploits the relationships among the problems of the GVRP family. Thus, he proposes a hierarchical taxonomy that classifies GVRP types into three

“floors” (see Figure 1), where the first floor represents the family of TSP problems. With the addition of vehicle capacity constraints, one transitions to the second floor of VRP problems, while precedence constraints define the third floor of pickup and delivery problems (PDP).

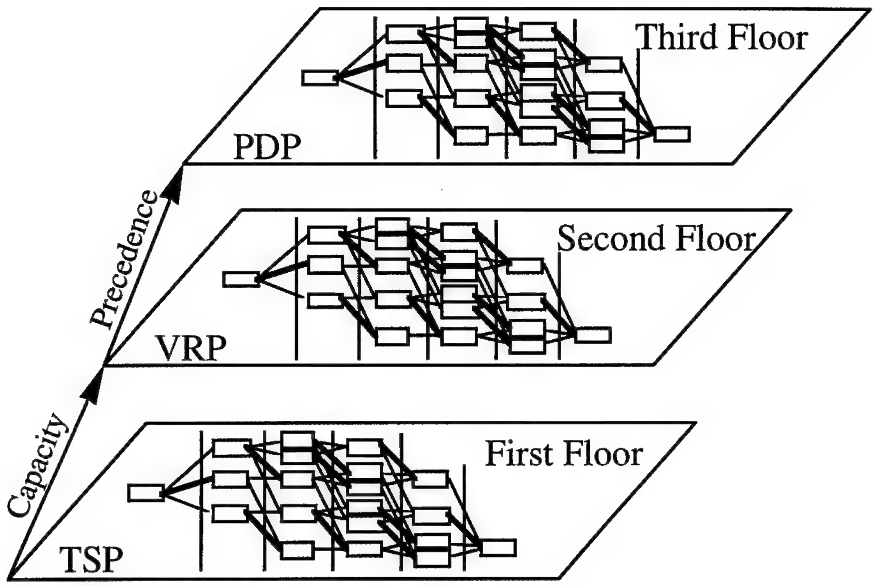


Figure 1. GVRP hierarchical classification scheme (Carlton 1995).

Each floor includes the following cases and their possible combinations:

1. SV: Single vehicle.
2. MVH: Multiple homogenous vehicles.
3. MV$\overline{H}$: Multiple non-homogenous vehicles.
4. SD: Single depot.
5. MD: Multiple depots.
6. TW: Time window constraints present.
7. RL: Route length constraints present.

In reference to problems on the first (TSP) floor alone, three works considered fundamental to the study of the GVRP acknowledge the failure of exact algorithms such as

branch-and-bound and dynamic programming to efficiently solve large instances. In addition to their remarks on problem complexity, Garfinkel (1985) steps through a transformation of the multiple salesman TSP (mTSP) to the TSP, while Christofides (1985) moves beyond optimization algorithms to discuss heuristics. Nemhauser and Wolsey (1995) provide a motivational example with visual steps to the optimal solution.

The literature identifies TS as a powerful heuristic for the GVRP. Laporte (1992b) surveys the exact and heuristic algorithms for the VRP, giving the highest marks to TS. Potvin, Kervahut, Garcia, and Rousseau (1996) compare the performance of their TS heuristic to that of five other documented heuristics upon the well-known Solomon datasets. Their TS employs a tabu list of fixed length and infeasible regions were not accessible to the search, yet the quality of solutions reached by their version of TS outperforms the other heuristics considered except a genetic sectoring algorithm called GIDEON (Thangiah 1993). It is in this context that we are motivated in applying TS to solving UAV routing problems.

Glover (1990a) provides guidelines in building a TS heuristic. In his fifth guideline, Glover strongly emphasizes using empirical results to improve move evaluations. His sixth guideline suggests the use of a frequency-derived (the frequency of revisited solutions) penalty to encourage diversification. With their reactive tabu search (RTS), Battiti and Tecchiolli (1994) extend Glover's sixth guideline by showing RTS to be a far more robust procedure than fixed and strict tabu search heuristics.

Battiti (1996) demonstrates how RTS effectively overcomes the drawbacks of computationally expensive parameter tuning and defeats the confinement of the search to local optima so common to fixed or strict implementations of tabu search. As a counter-example, Rochat and Taillard (1995) rely heavily on randomization to overcome these weaknesses by

generating a set of "good" simple TS solutions and then improving this set through a method reminiscent of genetic algorithms.

Carlton (1995) demonstrates how RTS obviates the need for any pre-processing of the sort required for Glover's target analysis. He also shows the randomization techniques proposed by Rochat and Taillard are unnecessary. The de-emphasis of randomization is a central tenet of tabu search (Glover 1993, 1997); and, although it requires greater computational effort, Carlton's wholly deterministic RTS implementation with an arbitrarily chosen initial solution consistently finds solutions of equal quality to those reached from feasible starting tours. By comparing his results to heuristics similar to the group compiled by Kervahut, Garcia, and Rousseau (1996), Carlton concludes the RTS dominates the others in solution quality and run time, including GIDEON (Thangiah 1993).

While the description of VRP's by Hall and Partyka (1997) captures most commercial applications, it falls short of military scenarios since it does not include the inherent variability of the operational environment's parameters in areas such as weather and vehicle survivability. Sisson (1997) investigates a military application, by applying Carlton's RTS to a unique mTSPTW formulation that incorporates the probabilities of vehicle attrition due to hostile forces into the objective function.

Jaillet and Odoni (1988) demonstrate the added complexity of a probabilistic TSP (PTSP) over the TSP. For even a simple heuristic like the nearest-neighbor, they find the computational effort increases by $O(n^2)$ over the deterministic TSP. Furthermore, their formulation of the PTSP is simple in comparison to the UAV problem since they only consider the probability that customers are not present. Jaillet and Odoni seek "well-behaved" or robust routes, but their

stochastic programming methods are bound to smaller numbers of customers by the necessary computational effort.

2.4. Methodology

As demonstrated by Kassou and Pecuchet (1994), object-oriented programming languages facilitate the inheritance and reuse of existing object definitions and methods. Our research makes full use of this approach by using CACI's object-oriented language MODSIM (CACI 1996a, 1996b; Marti 1997). In MODSIM, an object contains its own fields and routines (methods). While the contents of an object's fields can only be modified by its own methods, it can share those values with any other part of the program. Through inheritance, new object types arise from existing types by inheriting the fields and objects of the existing type. New objects can then redefine (or override) the inherited methods to behave differently, as well as add original fields and methods. For example, MODSIM library contains paired "definition" and "implementation" modules. The definition module contains the type declarations, while the implementation module contains the performing code. While a "main" module is similar to an implementation module in that it also contains executable code, it is used primarily to draw from existing libraries and is necessary for compilation into an executable file.

This research employs Carlton's taxonomy as the framework for applying the concept of inheritance between MODSIM optimization objects. Specifically, MODSIM objects accompanying this thesis correspond to the mTSPTW and the UAV version of the mTSPTW. As depicted in Figure 2, the transitions between provide an inheritance framework for building specialized RTS objects from the existing mTSPTW object. This structure allows us to quickly create customized heuristic-based objects for problems within the GVRP class.

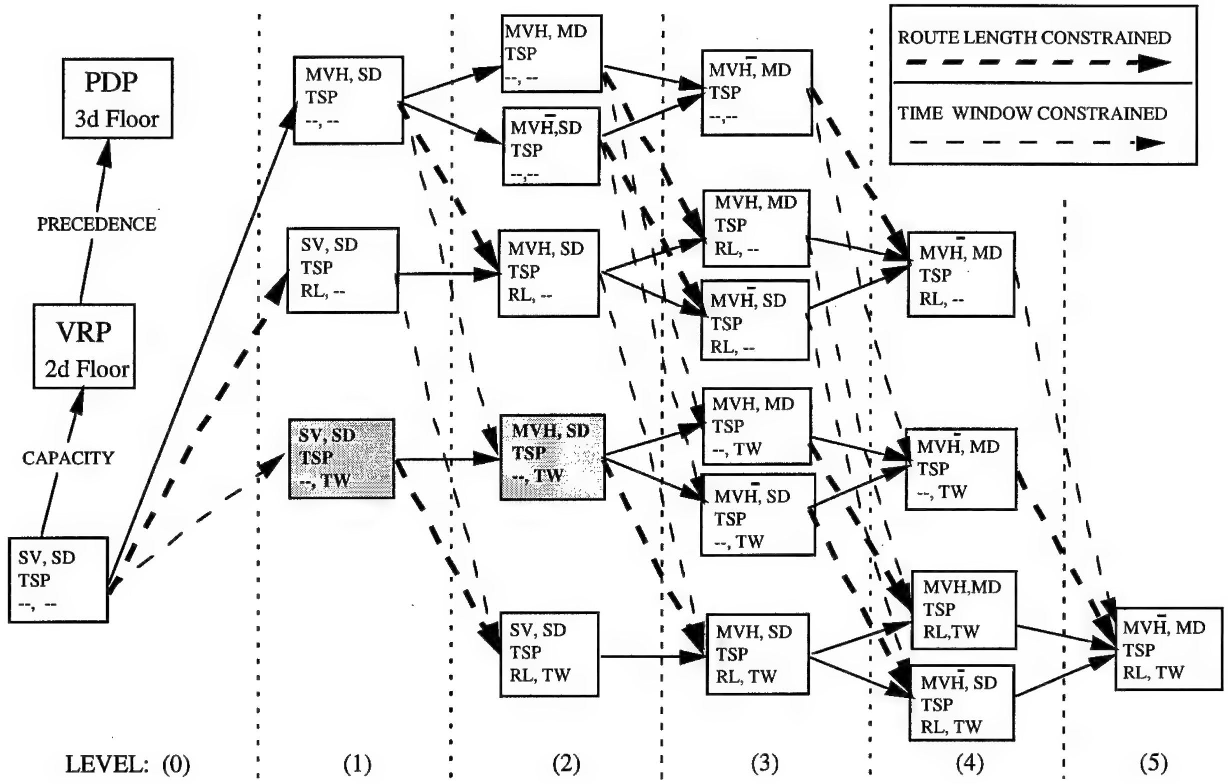


Figure 2. Traveling salesman problem hierarchy; GVRP first floor (Carlton 1995).

(Label format: Single (SV) or multiple (MVH) vehicles, single (SD) or multiple (MD) depots, traveling salesman problem (TSP), route length (RL) constrained, and time window (TW) constrained.)

Though possible in previous programming forms, the code encapsulation enforced and encouraged by MODSIM is useful in studying the GVRP. For instance, Carlton (1995) compared the results of different objective functions for the VRPTW. Strict encapsulation of code allows different objective functions to be efficiently introduced to the RTS object.

We created our RTS solver for the mTSPTW by translating Carlton's (1995) C language code into a set of MODSIM libraries and objects. These objects provide a "core" solver for the mTSP and mTSPTW instances of the GVRP family. With very minor adjustments, VRPTW instances can be solved as well. Testing the solver first on mTSP problems and then on mTSPTW problems verified the methods in the MODSIM objects. We note that altering the

object slightly for the mTSP case by removing time window calculations would speed iteration times, but is not necessary for verification purposes.

Using the notation presented earlier and a presentation order like that in Figure 2, the RTS object can solve the TSP (SV, SD TSP --, --) and the mTSP (MVH, SD TSP, --, --) by reading in time window widths far exceeding the tour length of any feasible solution and setting every node's load quantity to zero. After stepping line-by-line through the translated code with a 4-city problem to ensure accuracy, we further verified the heuristic's capabilities with the TSP by comparing the results of our runs of a 10-city TSP (Moore 1997) of known optimal solution and a problem from Reinelts' TSPLIB (1991).

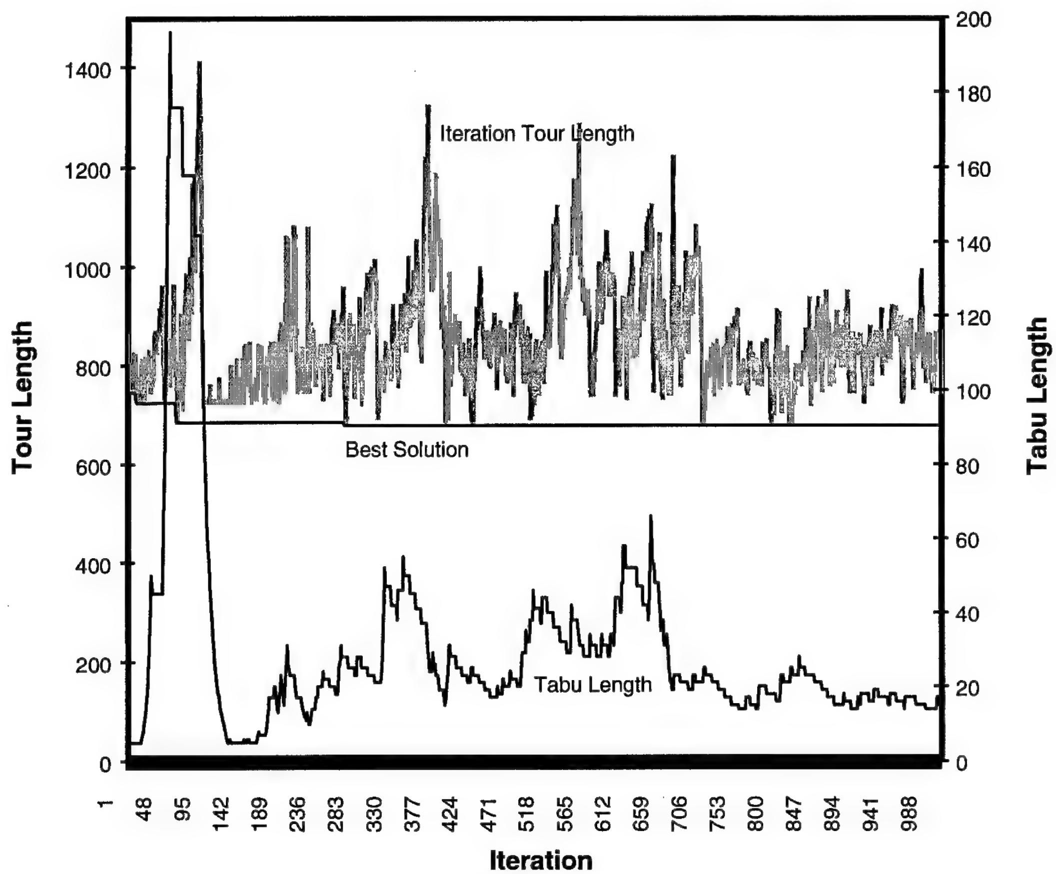


Figure 3. Ohio 10-City problem.

The peaks and valleys of the tabu length illustrate the ability of RTS to alternate from strategies of diversification to intensification, causing variation in the tour length. The optimal solution is found on iteration 288.

With any run of the RTS object, the analyst must decide the parameter settings beforehand. The parameters we consider are as follows:

1. PEN_{tw} is a multiplicative factor that weights the total time window infeasibility of all nodes, targets and vehicles. Because our formulation allows vehicles to wait, the early time window is considered soft; therefore, arriving early to create a wait time does not contribute to time window infeasibility. Only when we exceed the late time window do we consider the solution time window infeasible. If $PEN_{tw} = 1.0$, the weight on time window infeasibility is equal to all other portions of the tour.
2. DEPTH of search is the range of insertion moves considered for each node at each iteration. In all cases we use

   $$\text{DEPTH} = nc + nv - 2.$$

   The RTS logic avoids evaluating nonsensical moves regardless of the DEPTH chosen.
3. Tabu_length is the initial number of iterations a move attribute remains tabu. In all cases we use initialize Tabu_length to be

   Tabu_length = min(30, nc+nv-2).
4. INCREASE is the multiplicative factor used to increase the Tabu_length if a solution is revisited within the designated Cycle_length. In all cases we use

   INCREASE = 1.2.

5. DECREASE is the multiplicative factor used to decrease the Tabu_length if the steps since the last change to the Tabu_length exceed the moving average of Cycle_length. In all cases we use

$$\text{DECREASE} = 0.9.$$

6. Cycle_length is the minimum number of iterations between visitations to a solution that will not result in a Tabu_length increase. In all cases, we use

$$\text{Cycle\_length} = 50.$$

7. MINTL is the minimum to which Tabu_length can be decreased. In all cases, we use

$$\text{MINTL} = 5.$$

8. MAXTL is the maximum value to which Tabu_length can be increased. In all cases, we use

$$\text{MAXTL} = 2000.$$

We will adjust the value of PEN_{tw} for the UAV problem.

Not only can the code find solutions to the TSP, it can also find solutions for the TSPTW (SV, SD TSP --, TW) and the mTSPTW (MVH, SD TSP, --, TW). Carlton's work (1995) with Solomon's (1987) benchmark VRPTW datasets ignores the vehicle capacity and customer loads, thus providing the means to validate the performance of our RTS object within the TSPTW class (Table 1). Like Carlton, the PEN_{tw} weight is set to 1.0.

Table 1. TSPTW results.

Problem Name	Solution	# Vehicles Used	Iteration Best Solution Found	Time Best Solution Found
(5 vehicles available, 1000 iterations performed)				
C101	2441.3	3	23	5 secs
C102	2440.3	3	153	42
C103	2436.9	3	77	25
C104	2441.3	3	611	237
C105	2441.3	3	195	51
C106	2441.3	3	26	5
C107	2441.3	3	28	6
C108	2441.3	3	489	138
C109	2441.3	3	190	58
(10 vehicles available, 1000 iterations performed)				
R101	867.1	8	144	36
R102	797.1	7	34	9
R103	704.6	5	135	47
R104	666.9	4	85	33
R105	780.5	6	94	25
R106*	721.1	5	31	9
R107**	674.3	4	871	343
R108	647.3	4	58	23
R109	691.3	5	32	9
(10 vehicles available, 1000 iterations performed)				
RC101	711.1	4	341	87
RC102	601.7	3	20	6
RC103	583.0	3	226	78
RC104**	556.6	3	466	180
RC105	661.2	4	145	38
RC106	595.5	3	144	40
RC107	548.8	3	27	9
RC108	544.5	3	675	284

*Carlton found a better solution (715.4) on his 1209th iteration.
**Optimal solution found. Carlton did not find the optimal for these instances.

As expected, the iterations required to find solutions of a quality equal to those found by Carlton do not differ significantly from Carlton's results; what little differences that do exist are attributable to minute interference. However, the processing time required represents an order of magnitude increase over those resulting from the execution of Carlton's C code, despite the use of comparable systems. This comparison was made after running the original C code on an IBM compatible 486 with the original Pentium processor and running the MODSIM code on a Sun

Ultra 1. This increase appears to result from the simplistic form of C into which the MODSIM compiler translates the MODSIM code.

2.5. Implementation

2.5.1 Object Oriented Programming

Table 2 depicts the MODSIM structure of libraries and objects designed to solve mTSPTW problems. The pseudocode corresponds to the OBJECT, METHOD, and PROCEDURE columns in a hierarchical fashion similar to a path name. The heading ("main") indicates the implementation code can be found in the main module. In all cases, one follows the path to find the physical location of the code in the right-most nonblank space. If the code is not in main, the library listed refers to the library in which the right-most nonblank identifier lies. Dark gray spaces indicate that depth in the hierarchy is unneeded to specify the location.

The libraries provide a general framework for categorizing code into areas of similarity. Here, “tabuMod” contains code for use in GVRP-related tabu heuristics. The modules of “tsptwMod” contain code tailored for the mTSPTW problem, and “hashMod” holds the code for the creation and use of the hashing structure. As noted by Carlton (1995), many different objective functions can be used for GVRP problems, so “bestSolnMod” separates the code determining the best solution visited.

Table 2. Main module diagram, mTSPTW.

mTSPTW Reactive Tabu Search Pseudocode	SOURCE	OBJECT	METHOD	PROCEDURE
		Mtsptw (ModSim main module)		
0. Intialize: Structures, vectors, parameters	(main)			
1. Input problem instance:	tsptwMod	timeMatrix	readCarlton	
a. Number of iterations = niters	(main)			
b. Compute time/distance matrix	tsptwMod	timeMatrix	timeMatrix	
2. Select the starting tour	tsptwMod	startTour	startTour	
a. Compute initial schedule	tabuMod	"	"	tourSched
b. Compute initial tour penalties	tabuMod	"	startPenBest	compPens
c. Given penalties, compute initial tour cost	tabuMod	"	"	tsptwPen
d. Compute the initial hashing values: f(T) and thv(T)	tabuMod	"	"	tourHVwz
e. Save as initial best solution	bestSolnMod	"	"	twBestTT
3. While (k <= niters)	tsptwMod	reacTabuObj	search	
a. Look for the incumbent tour in the hashing structure	hashMod	"	"	lookfor
1) If found, update the iteration when found, and increase the tabu length, if applicable	tabuMod	"	"	cycle
2) If not found, add to the hashing structure, and decrease the tabu length, if applicable	tabuMod	"	"	nocycle
b. Perform "later" insertions: I(i,d) for i = 1 to n-1, d >= 1	tabuMod	reacTabuObj	search	SwapNode
1) Calculate the penalties associated with an insertion	tabuMod	"	"	compPens
2) Calculate the value of making this insertion	tabuMod	"	"	moveValTT
c. Evaluate all "earlier" insertions: I(i,d) for i = 3 to n, d <= -2	"	"	"	"
d. Move to the non-tabu neighbor according to an appropriate decision criteria. If all tours are tabu, move to the neighbor with the smallest move value, and reduce the tabu length	tabuMod	reacTabuObj	search	insert
e. Update the search parameters:				
1) Incumbent tour schedule	tabuMod	"	"	tourSched
2) Incumbent tour hashing value	tabuMod	"	"	tourHVwz
3) Retain the best feasible solution found and the tour with the smallest tour cost regardless of feasiblity	bestSolnMod	"	"	twBestTT
f. Increase iteration count: k = k + 1	tsptwMod	reacTabuObj	search	
4. Output results	tabuMod			twLoadToFile

Directions: To find where a portion of the pseudocode is executed, one can read the OBJECT, METHOD, and PROCEDURE columns like a hierarchial path name. The heading "(main)" indicates the implementation code can be found in the main module. Dark gray spaces indicate that space's depth in the hiearchy is unneeded to specify the location and " indicates the reference is identical to the last entry above it.

Tailoring the mTSPTW code for the UAV problem is simple. The light gray boxes in Table 3 illustrate the changes necessary to include random winds and service times into the problem. A distance matrix is calculated before the time matrix, and the time matrix calculation also differs.

Table 4 depicts the MODSIM code and RTS pseudocode when threats are included and we seek to maximize coverage. Since the UAV object is essentially a mTSPTW object with an altered objective function, we take advantage of the inheritance and polymorphism qualities of MODSIM by substituting only those portions of our mTSPTW associated with the move evaluation. Except for the lightly grayed-in areas, Table 4 is identical to the mTSPTW code in Table 2.

Table 3. Main module diagram, mTSPTW with winds.

mTSPTW (Winds Included) RTS Pseudocode	SOURCE	OBJECT	METHOD	PROCEDURE
		MUAV (ModSim main module)		
0. Intialize: Structures, vectors, parameters	(main)			
1. Input problem instance:	tsptwMod	timeMatrix	readProblem	
a. Number of iterations = niters	(main)			
b. Compute 'no wind' distance matrix	UAVMod	timeMatrix	distMatrix	
c. Compute the time matrix with winds	UAVMod	timeMatrix	timeMatrix	
3. Select the starting tour	tsptwMod	startTour	startTour	
a. Compute initial schedule	tabuMod	"	"	tourSched
b. Compute initial tour penalties	tabuMod	"	startPenBest	compPens
c. Given penalties, compute initial tour cost	tabuMod	"	"	tsptwPen
d. Compute the initial hashing values: f(T) and thv(T)	tabuMod	"	"	tourHVwz
e. Save as initial best solution	bestSolnMod	"	"	twBestTT
4. While (k <= niters)	tsptwMod	reacTabuObj	search	
a. Look for the incumbent tour in the hashing structure	hashMod	"	"	lookfor
1) If found, update the iteration when found, and increase the tabu length, if applicable	tabuMod	"	"	cycle
2) If not found, add to the hashing structure, and decrease the tabu length, if applicable	tabuMod	"	"	nocycle
b. Perform "later" insertions: I(i,d) for i = 1 to n-1, d >= 1	tabuMod	reacTabuObj	search	SwapNode
1) Calculate the penalties associated with an insertion	tabuMod	"	"	compPens
2) Calculate the value of making this insertion	tabuMod	"	"	moveValTT
c. Evaluate all "earlier" insertions: I(i,d) for i = 3 to n, d <= -2	"	"	"	"
d. Move to the non-tabu neighbor according to an appropriate decision criteria. If all tours are tabu, move to the neighbor with the smallest move value, and reduce the tabu length	tsptwMod	reacTabuObj	search	insert
e. Update the search parameters:				
1) Incumbent tour schedule	tabuMod	"	"	tourSched
2) Incumbent tour hashing value	tabuMod	"	"	tourHVwz
3) Retain the best feasible solution found and the tour with the smallest tour cost regardless of feasiblity	tsptwMod	"	"	twBestTT
f. Increase iteration count: k = k + 1	tsptwMod	reacTabuObj	search	
5. Output results	tsptwMod			twLoadToFile

Directions: To find where a portion of the pseudocode is executed, one can read the OBJECT, METHOD, and PROCEDURE columns like a hierarchial path name. The heading "(main)" indicates the implementation code can be found in the main module. Dark gray spaces indicate that space's depth in the hiearchy is unneeded to specify the location and " indicates the reference is identical to the last entry above it. Light gray spaces identify code that differs from the original mTSPTW formulation.

Table 4. Main module diagram, UAV.

UAV Reactive Tabu Search Pseudocode	MUAV (ModSim main module) SOURCE	OBJECT	METHOD	PROCEDURE
0. Intialize: Structures, vectors, parameters	(main)			
1. Input problem instance:	tsptwMod	timeMatrix	readProblem	
a. Number of iterations = niters	(main)			
b. Compute 'no wind' distance matrix	UAVMod	timeMatrix	distMatrix	
c. Compute the time matrix with winds	UAVMod	timeMatrix	timeMatrix	
3. Select the starting tour	tsptwMod	startTour	startTour	
a. Compute initial schedule	tabuMod	"	"	tourSched
b. Compute initial tour penalties	tabuMod	"	startPenBest	compPens
c. Given penalties, compute initial tour cost	tabuMod	"	"	tsptwPen
d. Compute the initial hashing values: f(T) and thv(T)	tabuMod	"	"	tourHVwz
e. Save as initial best solution	bestSolnMod	"	"	twBestTT
4. While (k <= niters)	UAVMod	uavRTSobj	search	
a. Look for the incumbent tour in the hashing structure	hashMod	"	"	lookfor
1) If found, update the iteration when found, and increase the tabu length, if applicable	tabuMod	"	"	cycle
2) If not found, add to the hashing structure, and decrease the tabu length, if applicable	tabuMod	"	"	nocycle
b. Perform "later" insertions: I(i,d) for i = 1 to n-1, d >= 1	UAVMod	uavRTSobj	search	SwapNode
1) Calculate the penalties associated with an insertion	tabuMod	"	"	compPens
2) Calculate the value of making this insertion	UAVMod	"	"	expCvrg
c. Evaluate all "earlier" insertions: I(i,d) for i = 3 to n, d <= -2	"	"	"	"
d. Move to the non-tabu neighbor according to an appropriate decision criteria. If all tours are tabu, move to the neighbor with the smallest move value, and reduce the tabu length	tabuMod	uavRTSobj	search	insert
e. Update the search parameters:				
1) Incumbent tour schedule	tabuMod	"	"	tourSched
2) Incumbent tour hashing value	tabuMod	"	"	tourHVwz
3) Retain the best feasible solution found and the tour with the smallest tour cost regardless of feasiblity	bestSolnMod	"	"	twBestTT
f. Increase iteration count: k = k + 1	UAVMod	uavRTSobj	search	
5. Output results	UAVMod			twCvrgToFile

Directions: To find where a portion of the pseudocode is executed, one can read the OBJECT, METHOD, and PROCEDURE columns like a hierarchial path name. The heading "(main)" indicates the implementation code can be found in the main module. Dark gray spaces indicate that space's depth in the hiearchy is unneeded to specify the location and " indicates the reference is identical to the last entry above it. Light gray spaces identify code that differs from the original mTSPTW formulation.

2.5.2 Embedded Optimization

Using the "portable" quality of our UAV object, we embed the UAV object in a Monte-Carlo simulation that seeks to model the inherent variability of the operational environment's parameters. Here, the simulation model introduces random variation in wind magnitude and direction, survival and service times. From the simulation, we identify routes and UAV fleet sizes that persist throughout the simulation space. We present this format as an example of embedded optimization, and in this context we offer the following formal definition:

> Embedded optimization occurs whenever a recognized optimization or heuristic procedure is an event within a simulation that directly affects the state of the system.

At its most generalized application, our definition of embedded optimization can encompass any optimization algorithm within a discrete-event simulation. However, for this paper we restrict our focus to optimization of the GVRP. The universal utility of this approach is suggested by Hall and Partyka's (1997) survey of industrial applications of VRP and their observation that GVRP involves interdependent problems. For example, given stochastic inputs to a Hall and Partyka problem, a separate optimization routine for both the TSPTW and crew scheduling subproblems can be inserted within a simulation considering multiple combinations of the random inputs.

Despite the wealth of real-world application, examples of embedded optimization in the

literature are rare (Hall 1997). Kassou and Pecuchet (1994) apply embedded optimization to job shop scheduling, where their object-oriented programming application uses a sophisticated optimization framework with an extensive user interface. Using the optimization routines within a simulation to provide possible scheduling scenarios, the authors arrive at “guided rules” for choosing one of the three optimization techniques available and how to guide the search. Kassou and Pecuchet (1994) introduce a feedback loop between the optimization search and the simulation processes, but the nature of the information shared is ambiguously defined and the user must maintain interface in the loop (even to the point of being the “Generator of rules”).

Brown and Graves (1981) furnish an example that does not adhere to our definition of embedded optimization, when they use optimization routines to replace time-consuming manual operations for the routing decisions of a nation-wide fleet of petroleum tank trucks. Whereas Brown and Graves refer to their structure as “embedded optimization,” their work better exemplifies an application of optimization routines where none were used previously, and not the embedding of optimization routines as an event within a simulation. Conversely, Glover, Laguna, and Kelly (1996) provide a good example of embedded optimization in a simulation that calls upon Glover’s scatter search (1977) and tabu search heuristics to find near-optimal solutions. The simulation can be optimized by a neural-net “accelerator.”

It should be stated that embedded optimization is not a method of simulation optimization. As defined by Carson and Maria (1997), simulation optimization seeks the best input variable values among all possibilities without explicitly evaluating each input combination or choice. Embedded optimization seeks to improve analysis capabilities beyond those supplied by most current software (Hall and Partyka 1997), because the analyst can move beyond the

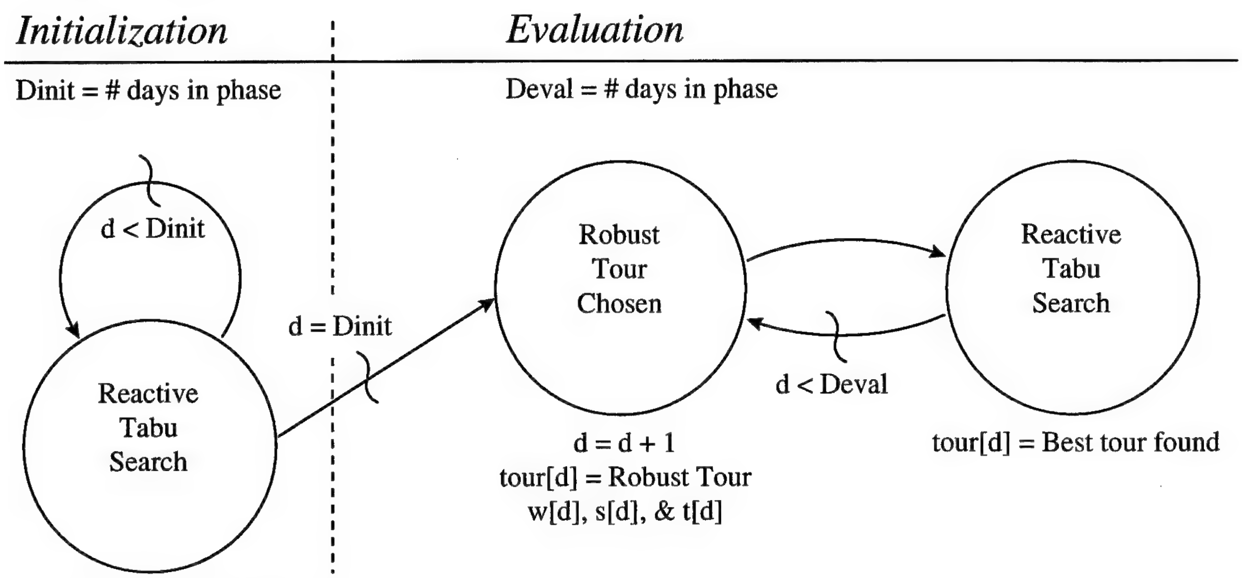
constraint of user-defined "what-if" situations to find robust answers. In our immediate application, we find routes robust to the variation of wind and threat inputs that lie outside the UAV operator's control, and therefore do not lend themselves to simulation optimization.

Glover (1977) introduces the concept of "strongly determined" and "consistent" variables, a distinction based on a continuum, not categorical, scale. His discussion focuses on the use of these concepts in the creation of integer programming heuristics. Our definition of robust is synonymous with his use of the term consistent; both are defined by the frequency of solution attributes that appear in a list of good quality solutions.

We make use of these concepts within a heuristic framework that seeks consistent answers to the UAV version of the mTSPTW. Using constructs similar to Schruben's (1993) event diagrams, Figure 4 illustrates our embedded optimization method for solving the UAV formulation. (Circles represent events, or state transitions, of the simulation, while arcs correspond to the scheduling of other events.)

Our simulation consists of two distinct phases: *initialization* and *evaluation*. In both phases, each replication of the simulation represents a 24-hour scenario whose random components (wind, survival probabilities, and service times) remain constant throughout that period (as drawn from their respective probability distribution functions). The purpose of the *initialization phase* is to find an initial pattern of routes that are robust. The *evaluation phase* determines the expected gain in performance of the RTS from the input of a robust solution.

State Variables:
d = Scenario, or day, counter.
w = Array of wind parameters: magnitude &direction.
s = Array of service times.
t = Array of threats, or probabilities of survival.
tour =Array of the current tour.



d = d + 1 (Day advances)
tour[d] = tour[d-1] (Search begins from previous day)
w[d], s[d], & t[d] (New scenario generated)
tour[d] = Best tour found (Search complete)

Figure 4. UAV Embedded Optimization Model.

The *initialization phase* of the simulation begins from an arbitrary solution. During this phase, when the TS for a scenario ends (i.e., the specified number of iterations are complete), new realizations of the random variables are generated for the next scenario. The TS then begins from the best solution of the just-completed scenario; in this manner the previous day's solution serves as a naïve forecast of the next day's solution. When the search completes the last scenario of the initialization period, the frequencies of routes used in the feasible solutions are summed in a route frequency matrix. The feasible solution whose routes are most persistent (i.e., those whose sum of route frequencies is the greatest) is termed the "Robust Tour."

The simulation then advances to the *evaluation phase* where the "Robust Tour" initializes the TS for each replication. Throughout this phase of the simulation, the route frequency matrix is updated. The continued update provides insight into what deviations from the robust tour will occur and allows the Robust Tour to be reformed by the results of a larger set of solution choices. Our approach is motivated by the hypothesis that in this context an RTS beginning from a Robust Tour will require fewer iterations to find good quality solutions (or better solutions may be found) than an RTS that begins from a naïve initial solution.

We note that this approach can be employed in a real time setting. Knowing the operating environment's past conditions, an *initialization phase* can provide a Robust Tour; then, once given the environment is known, a UAV operator could feed the Robust Tour into the RTS to determine routing assignments.

2.6. Results and Conclusions

2.6.1 Elucidation of Robustness

The first scenario set we analyze is a modification of Sisson's (1997) notional Nari dataset (Table 5). The coordinates are stated in miles from a fixed point. As we see from the time windows, the Nari dataset is essentially a TSP with a route length constraint of 24 hours. The services times are now stochastic and significantly shorter than the original with units also stated in hours. A range of service times is possible at each target.

In our model, the minimum service time is chosen unless a uniform random draw between 0 and 1 results in a value less than a predetermined probability (P_l) that the UAV may need to loiter at the target. If the first draw determines that an extended service time is required,

a second uniform random draw between the minimum and maximum service times determines the amount of time the UAV will loiter over the target.

The mean expected probabilities of survival (P_s) for the target nodes are set to either 0.8 or 0.9. In the UAV main module (Table 4) a uniform random draw determines if the *Ps* for any target changes between -0.1, 0.0, or 0.1. Each increment of change has a one-third chance of occurring. These levels are arbitrarily chosen to demonstrate the effect an objective of maximum coverage induces upon routing decisions within a stochastic threat environment.

Table 5. Nari dataset.

	Coordinates (in miles)		Early	Late	Service Time		Probability
	X	Y	Arrival	Arrival	Ranges (in hours)		of Survival
0*	100.286	64.286	0	24	0	0	1
1	7.714	381.429	0	24	1	5	0.9
2	55.714	6	0	24	1	5	0.8
3	81.429	351.429	0	24	1	5	0.9
4	58.286	342.857	0	24	1	5	0.8
5	65.143	325.714	0	24	1	5	0.9
6	34.286	327.429	0	24	1	5	0.8
7	70.286	296.571	0	24	1	5	0.9
8	27.429	291.429	0	24	1	5	0.8
9	93.429	297.429	0	24	1	5	0.9
10	48	280.286	0	24	1	5	0.8
11	76.286	269.143	0	24	1	5	0.9
12	120	274.286	0	24	1	5	0.8
13	160.286	291.429	0	24	1	5	0.9
14	100.286	251.143	0	24	1	5	0.8
15	114	216	0	24	1	5	0.9
16	205.714	234	0	24	1	5	0.8
17	104.571	219.429	0	24	1	5	0.9
18	144	220.286	0	24	1	5	0.8
19	126.857	203.143	0	24	1	5	0.9
20	231.429	217.714	0	24	1	5	0.8
21	292.286	191.143	0	24	1	5	0.9
22	181.714	145.714	0	24	1	5	0.8
23	200.571	140.571	0	24	1	5	0.9
24	291.429	137.143	0	24	1	5	0.8
25	214.286	121.714	0	24	1	5	0.9
26	248.571	92.571	0	24	1	5	0.8
27	274.286	82.286	0	24	1	5	0.9
28	291.429	78.857	0	24	1	5	0.8
29	332.571	82.286	0	24	1	5	0.9
30	349.714	80.571	0	24	1	5	0.8
31	377.143	84	0	24	1	5	0.9
32	375.429	99.429	0	24	1	5	0.8
33	385.714	111.429	0	24	1	5	0.9
34	402.857	115.714	0	24	1	5	0.8
35	404.571	106.286	0	24	1	5	0.9
36	396	94.286	0	24	1	5	0.8
37	432	92.571	0	24	1	5	0.9
38	437.143	70.286	0	24	1	5	0.8
39	447.429	43.714	0	24	1	5	0.9
40	472.286	33.429	0	24	1	5	0.8

* Denotes the depot.

Our *initialization phase* performs a 21-day simulation of the Nari scenario, where the winds vary between 205 and 245 degrees in origin at a magnitude ranging between 0 to 20 knots. Within the simulation, each target node is given a 0.5 probability (P_l) of its service (i.e., loiter) time increasing above its minimum level, while eleven vehicles are available for use. The RTS runs included 500 iterations and PEN_{tw} is set to 10.0 due to the results of test runs with the minimum travel time objective. Those test runs reveal a tendency of the RTS to choose time window infeasible solutions with a lower travel time over feasible solutions using more vehicles but incurring a higher travel time. Although we use the maximum coverage objective function for the Nari dataset, PEN_{tw} remains at 10.0 for all UAV scenarios of the research.

Before interpreting the route frequency matrix, we review the representation of the solution vector starting with a notional example in Figure 5.

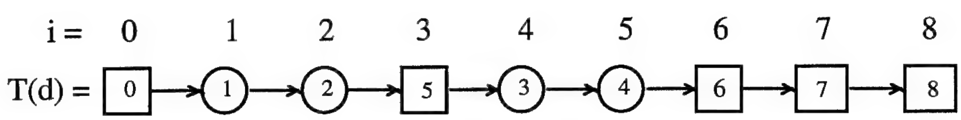


Figure 5. Notional tour array.

The node filling position i in the array is denoted inside the geometric figures. The figure shape corresponds to the type of node; circles represent targets while squares denote vehicles. Nodes 0 and 8 will never move since they are the depot; however, node 0 is considered a "vehicle" node. Using this format, the RTS essentially changes the order of this tour array in search of the optimal solution. Although not shown in Figure 5, each node contains a record of its position within the tour by carrying time window, arrival, departure, and wait time fields. In this way, the array provides a complete description of the routing and vehicle usage. For example, Figure 5 shows two vehicles are used in the vehicle-to-customer transitions of nodes 0 and 1 and nodes 5

and 3. We can tell four vehicles are available for use by counting vehicle nodes starting at node 0 but not including node 8. The arrival time to node 5 represents the tour length of the first vehicle's tour, while node 6 stores the tour length of the second. Nodes 6 and 7 represent unused vehicles since there are no intervening targets between them and node 8, the terminal depot.

Figure 6 displays the route frequency matrix resulting from the *initialization phase*. Where the row labels represent a departure node and the column labels an arrival node, the elements in the table represent the number of days that particular routing segment is an attribute of the solution tour within the 21-day simulation. Each row and column sums to 21 as every node appears in every tour array. The lightly grayed-in areas represent vehicle arcs.

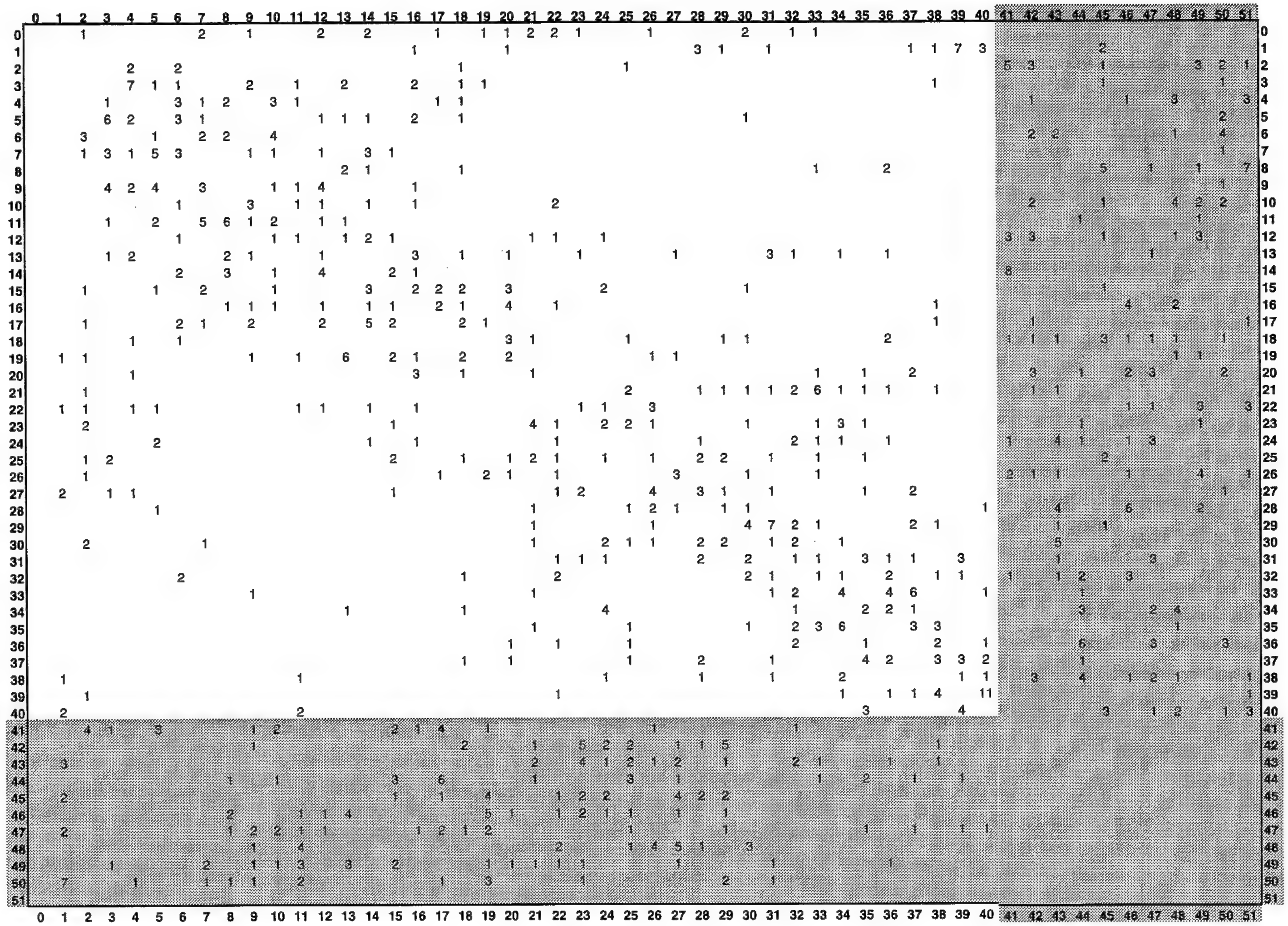
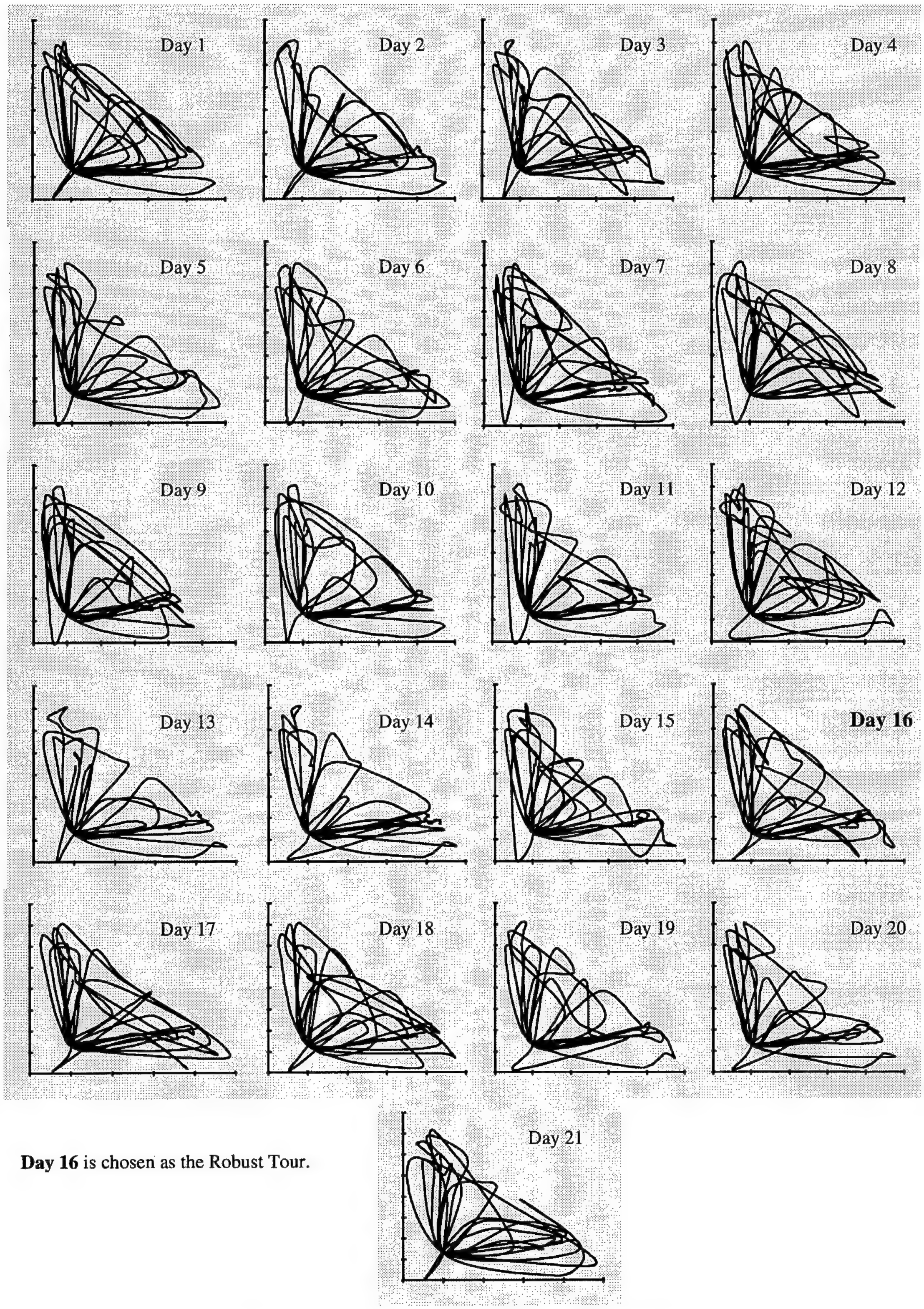


Figure 6. Nari scenarios, initialization route frequency.

Although some routes appear in roughly one third of the 21 random scenarios and the arc 39-to-40 appears in 11, distinguishing robust route structures is not easily done. Since this is a TSP problem, no time window restrictions are present to induce an order to the tour. Also, the P_s adjustments add considerable variation in what the objective function desires for an optimal tour order. One can see that all eleven vehicles are used in every scenario, as no vehicle-to-vehicle arcs are present. This result is sensible given the objective function seeks to maximize coverage.

Figure 7 graphically depicts the tours chosen for each day of the *initialization phase*. Again, the shapes of the tours do not readily yield to a visual examination. Based on the frequency its routes appear in the different tours chosen, the result of Day 16 is picked as the robust tour structure. Here, the depot lies above the first tick mark on the horizontal axis, where we see the eleven vehicle tours converge.



Day 16 is chosen as the Robust Tour.

Figure 7. Tours chosen for Nari scenarios.

Table 6 lists the results of both the *initialization* and *evaluation phases*. Inputs in the *evaluation phase* were identical to the *initialization phase*; however, the seeds to the random generators were changed. In short, the input of the chosen robust tour does not decrease iterations to the best found solution, not does it generate better quality solutions. Looking at the inputs, no relationship exists between the number of iterations required and the wind parameters or sum of service time increases. Although the wind inputs do generate variation in the tour structure, the effect of service time increases appears tied to the targets chosen for increase, not their sum. The *evaluation phase* changes the choice of Robust Tour from Day 5 of the *initialization* to Day 2, indicating a slight change in the average shape of the chosen tours when both phases are considered.

Table 6. Nari results, Initialization vs. Evaluation Phases.

Initialization

Day	Coverage	Travel Time	Iterations to Best	Wind Magnitude	Wind Origin**	Service Increase
1*	29.8	158.2	57	19	212	34.5
2	**32.8**	**172.2**	**127**	**3**	**212**	**47.3**
3	30.3	171.2	14	5	240	30.3
4	30.7	170.3	46	20	206	40.3
5	**31.3**	**168.0**	**15**	**5**	**216**	**29.4**
6	31.2	189.9	14	17	222	52.8
7	30.6	199.8	15	13	238	50.9
8	31.1	177.1	18	17	242	40.8
9	19.8	178.9	73	8	229	41.5
10	31.0	175.1	12	4	224	28.3
11	29.7	190.6	62	9	226	57.1
12	31.7	190.4	27	5	230	60.4
13	31.9	168.7	26	17	229	41.2
14	32.2	175.0	46	17	228	46.0
15	30.0	185.5	55	18	235	45.0
16	31.8	171.0	27	18	223	43.9
17	31.2	190.1	118	9	226	58.2
18	31.0	180.9	21	10	221	44.5
19	32.1	186.1	54	11	221	42.0
20	32.0	172.8	24	15	241	42.1
21	31.2	183.7	35	9	234	48.8
Means:	30.7	179.9	41.5	11.5	227	44.5
Std Dev:	2.67	9.15	33.0	5.5	9.55	8.87

Day 5 chosen as Robust Tour.

*Not used in calculation of means as it begins from an arbitrary solution.

** Cartesian coordinate system.

Evaluation

Day	Coverage	Travel Time	Iterations to Best	Wind Magnitude	Wind Origin**	Service Increase
22	31.2	189.9	34	9	211	40.3
23	29.7	203.9	37	9	235	48.5
24	29.7	211.3	22	13	221	47.7
25	30.8	200.9	24	10	241	32.7
26	31.6	174.2	41	0	224	32.7
27	30.5	187.0	33	9	217	36.6
28	31.2	206.4	25	7	225	40.3
29	30.2	178.9	40	11	241	30.0
30	32.1	170.8	31	8	237	25.2
31	31.7	197.5	151	4	222	40.3
32	30.0	200.9	28	11	212	55.2
33	28.7	176.3	32	1	210	27.5
34	32.5	185.7	114	17	223	58.5
35	32.3	174.1	33	6	213	28.7
36	31.3	222.3	26	5	214	59.2
37	29.9	203.9	22	1	245	36.0
38	**32.5**	177.8	**30**	**15**	**207**	**41.0**
39	**32.0**	181.1	**37**	**10**	**208**	**42.0**
40	**31.7**	169.8	**66**	**4**	**221**	**34.9**
41	**31.9**	192.9	**41**	**11**	**240**	**52.0**
Means:	31.1	190.3	43.4	8.1	223	40.5
Std Dev:	1.09	15.05	32.5	4.6	12.3641	10.20

On Days 38-41, Day 2 chosen as Robust Tour.

The second scenario set we analyze is a notional Bosnia dataset provided by the 11th Reconnaissance Squadron (Bergdahl 1998) where we are given stochastic service times (Table 7). In the operational setting, coordinates are specified in latitude and longitude. The time windows are specified in military time. A number of nodes must be visited twice, and the time windows of the second visit follow the service time ranges that apply to both visits.

Table 7. Notional Bosnia data.

	Latitude			Longitude			First Visit		Service Time		Second Visit	
Target Name	DEG	MIN	SEC	DEG	MIN	SEC	Early Arrival	Late Arrival	Ranges (in Min)		Early Arrival	Late Arrival
TASZAR HUNGARY, DEPOT	46	24	0	17	54	0						
CORRIDOR, SZULOK HUNGARY	46	3	45	17	32	44						
CORRIDOR, SRBAC BOSNIA	45	24	0	17	30	0						
DUMDVGA	44	58	29	16	50	34	1015	1500	30	180	1900	2300
MASTYE	44	58	46	16	38	56	1015	1500	30	180	1900	2300
GARRED AAA SITE	44	58	4	16	39	31	1015	1500	2	15	1900	2300
THARMET HEAVY WPN DEPOT	44	58	33	16	39	18	1015	1500	2	30	1900	2300
THARMET HEAVY WPN DEPOT	44	58	39	16	39	41	1015	1500	2	30	1900	2300
THARMET HEAVY WPN DEPOT	44	58	59	16	39	28	1015	1500	2	30	1900	2300
SERDONA COMM SITE	44	59	2	16	39	56	1015	1500	2	30	1900	2300
SERDONA COMM SITE	44	59	11	16	40	19	1015	1500	2	30	1900	2300
SERDONA COMM SITE	44	59	15	16	39	20	1015	1500	2	30	1900	2300
SUSPECTED WPN STORAGE	44	59	9	16	39	10	1015	1500	2	30	1900	2300
SUSPECTED WPN STORAGE	44	54	52	16	34	47	1015	1500	2	30	1900	2300
SUSPECTED WPN STORAGE	44	51	49	16	41	37	1015	1500	2	30	1900	2300
SUSPECTED WPN STORAGE	45	0	7	16	34	47	1015	1500	2	30	1900	2300
SUSPECTED WPN STORAGE	44	59	9	16	49	17	1015	1500	2	30	1900	2300
SUSPECTED WPN STORAGE	44	57	41	16	39	35	1015	1500	2	30	1900	2300
AIR DEFENSE, SAM, PROBABLE SA-2	44	57	23	16	51	45	1015	1500	2	30	1900	2300
AIR DEFENSE, SAM, PROBABLE SA-2	44	57	45	16	49	28	1015	1500	2	30	1900	2300
AIR DEFENSE, SAM, PROBABLE SA-2	44	55	57	16	43	52	1015	1500	2	30	1900	2300
AIR DEFENSE, SAM SITE RADAR	44	57	47	16	39	54	1015	1500	2	30	1900	2300
DROMADA HQ SITE	45	0	7	16	53	49	1015	1500	30	120	1900	2300
DROMADA WAREHOUSE	44	53	31	16	54	12	1015	1500	2	60	1900	2300
OMANSKI BARRACKS	44	45	34	17	10	34	1500	1715	5	120		
OMANSKI BARRACKS	44	48	19	17	12	14	1500	1715	5	120		
OMANSKI BARRACKS	44	51	2	17	13	24	1500	1715	5	120		
BOLSTAVEC TANK RALLY POINT	44	50	51	17	14	39	1500	1715	2	30		
BOLSTAVEC TANK RALLY POINT	44	56	17	17	17	41	1500	1715	2	30		
KRAJACHASTANE STORAGE BUNKER	44	55	51	17	17	51	1500	1715	2	30		
KRAJACHASTANE STORAGE BUNKER	44	56	7	17	18	23	1500	1715	2	30		
GOLPRTUNIY ROAD	44	28	13	17	1	18	1730	1830	20	40		
GOLPRTUNIY ROAD	44	27	29	17	1	46	1730	1830	20	40		
GOLPRTUNIY ROAD	44	27	10	17	2	24	1730	1830	20	40		

The nodes that must be visited twice are modeled as two independent nodes. The data set is clustered as target nodes separated into remote operating zones (ROZ) such that time windows between ROZ's do not overlap.

Figure 8 displays the route frequency matrix resulting from a 21-day simulation of the Bosnia scenario, where the winds vary between 265 and 315 degrees in origin at a magnitude ranging between 10 to 25 knots. Each target node is given a 0.3 probability of its service time increasing above its minimum level. If the random draw is not less than 0.3, the minimum service time applies. Five vehicles are available for use.

As the eye readily distinguishes, the matrix is sparse. Several distinct route segments emerge as being persistent throughout the simulation space. For example, segments 13-to-11, 11-to-12, 19-to-15, and 15-3 are present in the best solutions of all 20 replications. Further examination reveals that the vehicles denoted by node identification numbers 55 and 56 were never used. The sparseness of the matrix partly results from the clustering of nodes into ROZ's.

Similar to Figure 7, Figure 9 graphically depicts the tours of the Bosnia *initialization phase*. Longitude forms the horizontal axis, latitude the vertical. The depot lies in the top right hand corner of each chart.

Like Table 6, Table 8 lists the results by phase of the Bosnia dataset. Although the tour of Day 14 remains the robust tour throughout the *evaluation phase*, it fails to reduce the number of iterations to the best found feasible solutions. In fact, the number of iterations increases over the naïve input.

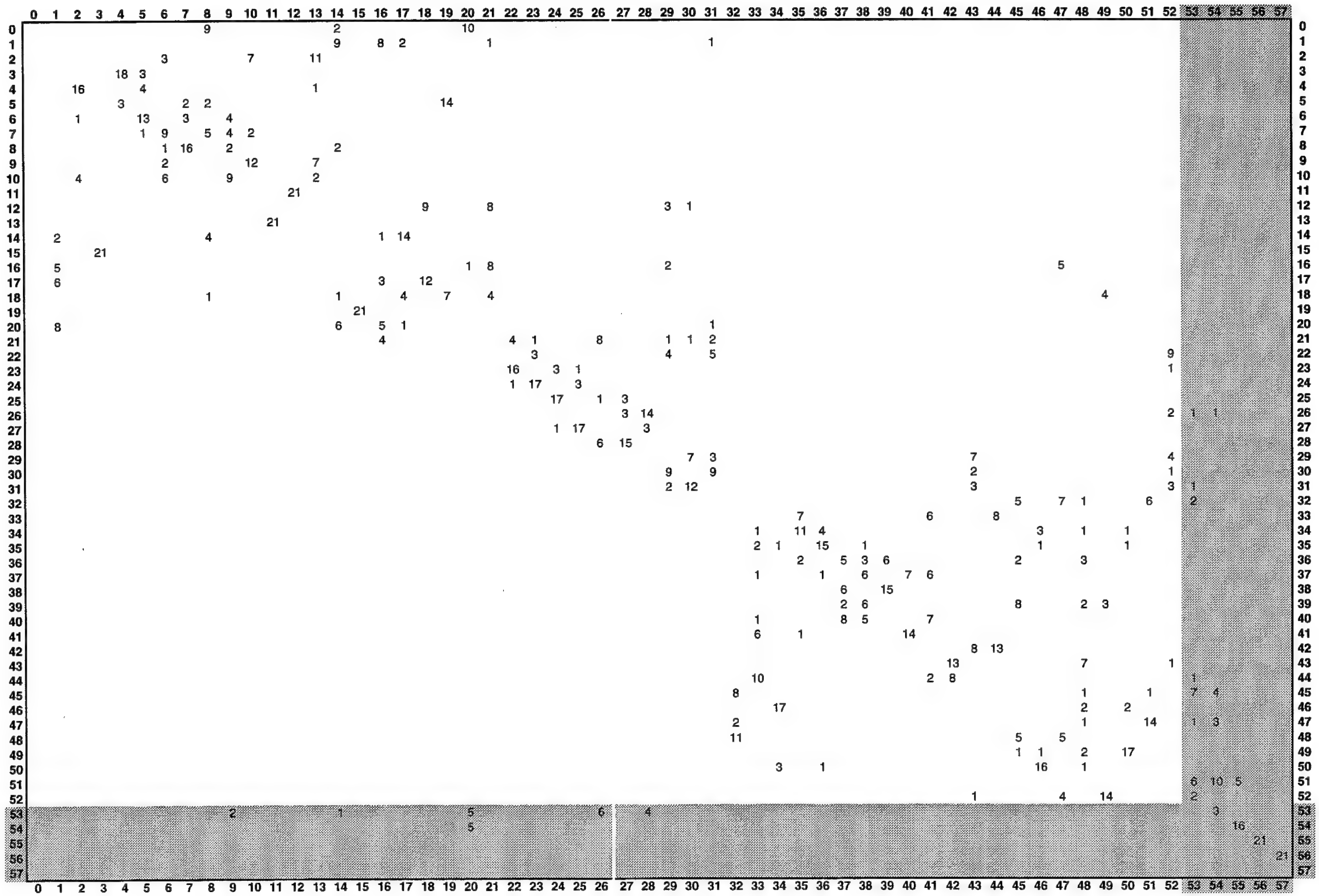


Figure 8. Bosnia scenario, initialization route frequency.

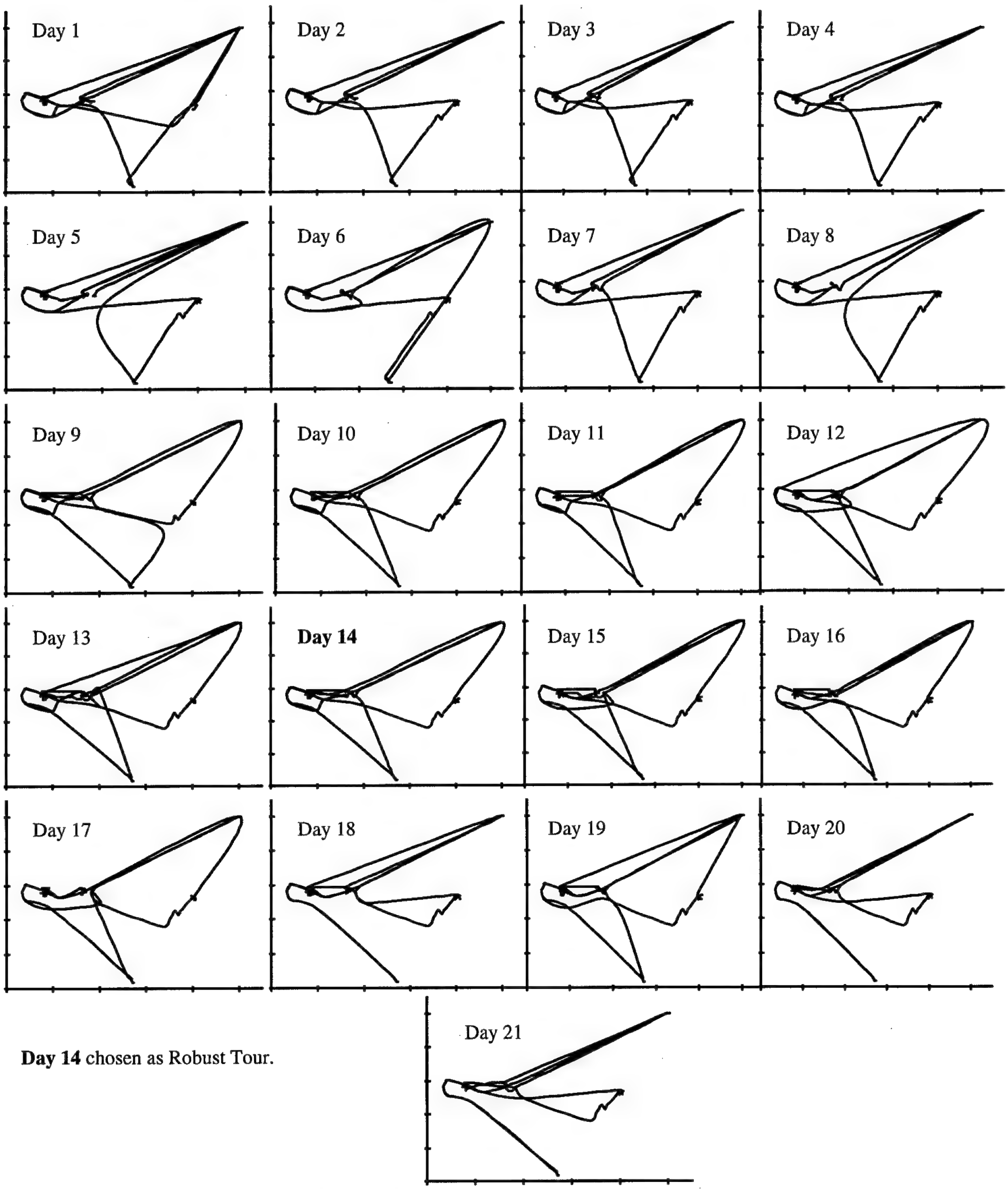


Figure 9. Tours chosen for Bosnia scenarios.

Table 8. Bosnia results, Initialization vs. Evaluation Phases.

	Initialization						Evaluation						
Day	Travel Time	Vehicles Used	Iterations to Best	Wind Magnitude	Wind Origin**	Service Increase	Day	Travel Time	Vehicles Used	Iterations to Best	Wind Magnitude	Wind Origin**	Service Increase
1*	19.21	3	94	11	273	8.17							
2	15.66	2	4	11	302	5.68	22	18.64	2	101	23	309	7.93
3	16.62	2	3	17	285	6.28	23	20.04	3	80	15	300	8.08
4	15.34	2	4	13	310	5.36	24	20.93	4	158	24	278	7.14
5	18.33	2	7	1	289	7.73	25	23.86	5	92	15	265	10.26
6	13.64	2	70	11	280	3.63	26	12.27	3	493	5	287	1.84
7	15.16	2	457	8	290	5.30	27	14.25	3	112	15	313	3.38
8	15.00	2	17	14	310	5.02	28	16.47	3	87	19	310	5.19
9	19.20	2	95	11	305	9.15	29	15.67	3	388	2	268	5.28
10	12.75	2	187	5	286	2.93	30	16.95	3	55	22	267	4.89
11	17.61	2	12	14	274	7.40	31	17.61	3	46	7	315	7.10
12	16.19	2	319	1	271	6.38	32	19.44	4	98	23	292	6.73
13	15.59	2	41	21	287	4.89	33	13.46	2	302	1	268	3.84
14	**14.50**	**2**	**52**	**8**	**275**	**4.56**	34	16.88	3	489	20	287	5.33
15	15.86	2	360	6	276	5.92	35	16.52	5	405	6	272	3.81
16	18.29	2	9	2	315	8.59	36	15.60	4	231	17	309	3.46
17	16.41	2	422	19	267	5.98	37	15.97	3	370	2	283	5.59
18	20.10	2	59	12	269	9.70	38	15.55	4	159	14	298	3.31
19	12.66	2	46	5	285	2.87	39	18.01	4	245	3	272	6.29
20	15.71	2	298	14	309	5.45	40	16.07	3	486	0	290	5.78
21	14.95	2	8	11	288	4.68	41	17.30	3	274	1	272	6.87
Means:	15.98	2	124	10	289	5.88		17.07	3.4	234	12	288	5.61
Std Dev:	1.97	0.0	155.9	5.7	15.1	1.88		2.64	0.8	157.1	8.7	17.2	2.02

Result of Day 14 chosen as the Robust Tour and remains so throughout the Evaluation.

* Not used in calculation of means as it begins from an arbitrary solution.

** Cartesian coordinate system.

From both scenarios, we surmise our definition of the robust tour does not significantly reduce iteration counts or generate better quality solutions to the search and we thereby reject our earlier stated hypothesis. This lack of iteration reduction in the *evaluation phases* for both scenarios speaks to the power of RTS; the RTS heuristic finds a good solution quickly enough to offset any advantage provided by a robust starting solution. Although Carlton (1995) reported better results when the RTS begins from a tour created by Solomon's respected insertion heuristic (1987), his RTS still produces solutions of a quality favorably comparable to most of the other published heuristics for the VRP.

2.6.2 Analysis of Capability

The 11th Reconnaissance Squadron (RS) operates only one UAV in the air at a time (Bergdahl 1998). While the 11th RS plans to increase the number of UAV's that can operate simultaneously, they do not know what this capability will achieve in the field. Using our embedded optimization model, we can evaluate mission capability as measured by the number of feasible solutions by generating random scenarios from the notional Bosnia dataset. Specifically, for a given combination of vehicle availability and probability of increased loiter time (P_l), we conduct twenty replications and observe the number of feasible tours. For every combination of vehicle availability and P_l, we use the same random number streams; thus, by repeating the same twenty scenarios we can directly compare the results.

Table 9. Number of feasible solutions in 20 replications.

		Probability of Loiter		
		0.1	0.3	0.5
Vehicles	1	11	3	0
	2	14	19	17
	3	20	20	20

As Table 9 shows, the current capability of one vehicle is woefully lacking given any strong increase in the probability of extensive loiter times. These results are intuitively understandable with one exception; the number of feasible solutions for two vehicles increases significantly when P_l increases from 0.1 to 0.3. A knowledge of our RTS logic explains the result.

As mentioned previously, the multiplicative factor PEN_{tw} weights time window infeasibility defined by

$$P_{tw}(T) = PEN_{tw} * \sum_{i=0}^{nc+nv} \max(0, A_i - l_i)$$

where $P_{tw}(T)$ is a portion of the minimum travel time objective function within the RTS. Violations of target time windows and vehicle route lengths are weighted equally regardless of the value of PEN_{tw} (vehicle route length constraints are modeled as time windows within the heuristic). As we increase PEN_{tw}, we decrease the probability the heuristic will choose time window infeasible solutions over feasible solutions of greater travel time. Increasing PEN_{tw} beyond its current value of 10.0 will eventually remove the nonsensical result and remove the artificial under-utilization of the second vehicle.

A more in-depth analysis of the tours can shed a full light on the operational capabilities of the vehicles within our notional scenario and clarify our discussion of increasing PEN_{tw}. Table 10 lists the result of Day 16 of the *initialization phase*. To get the actual travel time, the

"Tour Length" value listed must be divided by 100. Looking at the column of arrival times to each node (denoted "Arr") we see that the arrival to vehicle node 53 contains the tour length of the first vehicle, while the arrival to the terminal depot contains the tour length of the second vehicle. If the "Arr" column exceeds the late arrival time ("lArr") of the vehicle at the end of a vehicle's tour, then that tour is route length infeasible. The route length constraint is the only constraint violated on Day 16.

A wait time ("Wait") occurs in front of target nodes that are first visited in the vehicle tours because the vehicle departure is set equal to the early arrival ("eArr") time of the vehicle. We do not include this wait time in the sum of wait times reported for target nodes, since it is not necessary for a dispatcher to follow the heuristic's simplistic logic for the departure time ("Dep"). The "s" column contains the service time used on this day, while "slo" and "shi" specify respectively the minimum and maximum possible service times, respectively.

Table 10. Bosnia scenario, Day 16 of the initialization phase.

DAY 16 Search complete: BEST TOUR (NOT FEASIBLE)
WIND: magnitude = 2 direction (rads) = 5.4978
Tour Length: 22523

TYPE	ID	eArr	lArr	Arr	Dep	Wait	s	slo	shi
DEPOT	0	9.25	23.75	9.25	9.25	0	0	0	0
NODE	20	10.25	15	9.72	10.25	0.53	1.994	0.5	2
NODE	14	10.25	15	12.29	12.29	0	0.033	0.033	0.5
NODE	17	10.25	15	12.34	12.34	0	0.033	0.033	0.5
NODE	18	10.25	15	12.43	12.43	0	0.167	0.033	0.5
NODE	19	10.25	15	12.64	12.64	0	0.033	0.033	0.5
NODE	15	10.25	15	12.68	12.68	0	0.033	0.033	0.5
NODE	3	10.25	15	12.72	12.72	0	0.033	0.033	0.25
NODE	4	10.25	15	12.76	12.76	0	0.033	0.033	0.5
NODE	5	10.25	15	12.79	12.79	0	0.033	0.033	0.5
NODE	8	10.25	15	12.84	12.84	0	0.033	0.033	0.5
NODE	7	10.25	15	12.87	12.87	0	0.033	0.033	0.5
NODE	6	10.25	15	12.91	12.91	0	0.033	0.033	0.5
NODE	9	10.25	15	12.95	12.95	0	0.033	0.033	0.5
NODE	10	10.25	15	12.98	12.98	0	0.489	0.033	0.5
NODE	2	10.25	15	13.47	13.47	0	0.5	0.5	3
NODE	13	10.25	15	14.02	14.02	0	0.033	0.033	0.5
NODE	11	10.25	15	14.12	14.12	0	0.033	0.033	0.5
NODE	12	10.25	15	14.22	14.22	0	0.033	0.033	0.5
NODE	21	10.25	15	14.38	14.38	0	0.033	0.033	1
NODE	16	10.25	15	14.47	14.47	0	0.033	0.033	0.5
NODE	1	10.25	15	14.52	14.52	0	0.5	0.5	3
NODE	32	19	23	15.02	19	3.98	2.992	0.5	3
NODE	51	19	23	22.03	22.03	0	1.704	0.5	2
VHCL	53	9.25	23.75	24.21	9.25	0	0	0	0
NODE	26	15	17.25	9.64	15	5.36	0.033	0.033	0.5
NODE	28	15	17.25	15.04	15.04	0	0.033	0.033	0.5
NODE	27	15	17.25	15.08	15.08	0	0.114	0.033	0.5
NODE	25	15	17.25	15.27	15.27	0	0.407	0.033	0.5
NODE	24	15	17.25	15.69	15.69	0	0.083	0.083	2
NODE	23	15	17.25	15.81	15.81	0	0.66	0.083	2
NODE	22	15	17.25	16.51	16.51	0	0.083	0.083	2
NODE	31	17.5	18.5	16.85	17.5	0.65	0.333	0.333	0.666
NODE	30	17.5	18.5	17.84	17.84	0	0.625	0.333	0.666
NODE	29	17.5	18.5	18.48	18.48	0	0.333	0.333	0.666
NODE	52	19	23	19.16	19.16	0	0.085	0.033	1
NODE	43	19	23	19.37	19.37	0	0.398	0.033	0.5
NODE	42	19	23	19.84	19.84	0	0.033	0.033	0.5
NODE	44	19	23	19.95	19.95	0	0.033	0.033	0.5
NODE	33	19	23	20.02	20.02	0	0.5	0.5	3
NODE	35	19	23	20.52	20.52	0	0.118	0.033	0.5
NODE	34	19	23	20.65	20.65	0	0.033	0.033	0.25
NODE	46	19	23	20.69	20.69	0	0.064	0.033	0.5
NODE	50	19	23	20.75	20.75	0	0.033	0.033	0.5
NODE	36	19	23	20.8	20.8	0	0.033	0.033	0.5
NODE	37	19	23	20.83	20.83	0	0.033	0.033	0.5
NODE	41	19	23	20.87	20.87	0	0.033	0.033	0.5
NODE	40	19	23	20.91	20.91	0	0.033	0.033	0.5
NODE	38	19	23	20.94	20.94	0	0.363	0.033	0.5
NODE	39	19	23	21.31	21.31	0	0.455	0.033	0.5
NODE	49	19	23	21.82	21.82	0	0.033	0.033	0.5
NODE	45	19	23	21.92	21.92	0	0.033	0.033	0.5
NODE	48	19	23	21.97	21.97	0	0.235	0.033	0.5
NODE	47	19	23	22.22	22.22	0	0.033	0.033	0.5
DEPOT	54	9.25	23.75	22.77	9.25	0	0	0	0

Table 11 shows the mean waiting time ("Wait") for all targets visited after the first target of the vehicle tour(s) (a subset we will refer to as "interior targets") and the mean amount by which the route length constraint is exceeded ("Infeas") for the infeasible solutions of Table 9. If a vehicle tour is not infeasible, the tour length (with all wait times included) is listed in a light gray box in the "Infeas" column.

Table 11. Matrix of waiting times and route length infeasibility.

		Probability of Loiter					
		0.1		0.3		0.5	
Vehicles		Wait	Infeas	Wait	Infeas	Wait	Infeas
1	Veh 1	2.53	0.73	0.48	1.81		
2	Veh 1	3.24	0.83	3.98	0.46	1.94	0.53
	Veh 2	-	15.8	-	22.77	1.64	0.18
3	Veh 1						
	Veh 2						
	Veh 3						

Legend: Wait = Average sum of wait times for "interior" targets

Infeas = Average amount of route length violation

Route Length of the feasible vehicle

When the sum of interior wait times exceeds the amount of route length infeasibility, we know that if we relax the target time window constraints, the interior wait times will decrease. Therefore, we conclude that an infeasible answer whose mean sum of interior wait times exceeds its average amount of route length violations can become feasible by the increasing time window widths of the targets. If the time window constraints can be relaxed in this manner, then a single UAV can fly feasible tours when $P_l \leq 0.1$; and two can fly feasible tours for $P_l \leq 0.5$. By this logic, our experiments suggest that a feasible tour is not possible for one UAV when $P_l \geq 0.3$. The results for one vehicle and a P_l of 0.5 are not shown because all instances are infeasible; conversely, when three vehicles are available feasible tours occur when $P_l \leq 0.5$.

2.7. Recommendations for Further Study

Comparing alternatives such as those in section 2.6.2 is a classic application of simulation. With embedded optimization, comparisons are quickly made of a highly complex problem. A closer working relationship with UAV operators is likely to result in more opportunities for comparisons such as these.

Despite the failure of our chosen robust tour, plausible uses of embedded optimization to achieve iteration reduction remain. Instead of injecting random inputs into the RTS, an experimental design can be constructed to run the RTS for each design point. Given a set of inputs, a look-up table from the design could then serve to provide a better initial solution. If the inputs are not similar to any one design point, a method of path-relinking (Glover 1997) could be used to initiate the RTS.

Our work only scratches the surface of the applications available for embedded optimization, and additional opportunities abound for improvements to the simple model we present. Glover, Laguna, and Kelly (1996) contribute a possible improvement to the MODSIM objects created here. As their OptQuest software iterates through the simulation and optimization loop, a neural network "accelerator" may be called upon (at the user's discretion) to screen simulation input parameters that are likely to result in a poor overall measure of system performance (such as high cost). With a GVRP, infeasible scenarios could be screened and insight gained about the parameters making the solution space infeasible.

Finally, an important contribution from this effort is the MODSIM libraries. Using these libraries, future code can be quickly tailored to specific members of the GVRP family. Even if the programmer is not working within MODSIM, the libraries provide for a straightforward

translation given the "strongly typed" nature of MODSIM and the strict adherence to code encapsulation they embody. Their use can reduce the up-front coding time so often required for GVRP research.

Bibliography

Battiti, Roberto. "Reactive Search: Toward Self-Tuning Heuristics," *Modern Heuristic Search Methods*. Ed. V.J. Rayward-Smith, I.H. Osman, C.R. Reeves and G.D. Smith, New York: John Wiley and Sons, Inc., (1996).

Battiti, Roberto and Giampietro Tecchiolli. "The Reactive Tabu Search," *ORSA Journal on Computing*. 6 (2), 126-140 (1994).

Burkard, R. E., V. G. Deineko, R. van Dal, J. A. A. van der Veen, G. J. Woeginger. *Well-Solvable Special Cases of the TSP: A Survey*. Working Paper, SFB-Report 52. Graz University of Technology, Austria, (December 1995).

Carlton, William B. *A Tabu Search Approach to the General Vehicle Routing Problem*. Ph.D. dissertation. University of Texas, Austin, (1995).

Carson, Yolanda and Anu Maria. "Simulation Optimization: Methods and Applications," *Proceedings of the 1997 Winter Simulation Conference*. Ed. S. Andradottir, K. J. Healy, D. H. Withers, and B. L. Nelson. Atlanta GA, (7-10 December 1997).

Christofides, N. "Vehicle Routing," *The Traveling Salesman Problem*. Ed. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. New York: John Wiley & Sons, Inc., (1985).

Garfinkel, R. S. "Motivation and Modeling," *The Traveling Salesman Problem*. Ed. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. New York: John Wiley & Sons, Inc., (1985).

Glover, F. "Heuristics for integer programming using surrogate constraints," *Decision Sciences, 8*: 156-166 (1977).

Glover, F. "Tabu Search: A Tutorial," *Interfaces, 20*: 74-94 (July-August 1990a).

Glover, F. "Tabu Search - Part I," *ORSA Journal on Computing, 1*: 190-206 (Summer 1989).

Glover, F. "Tabu Search - Part II," *ORSA Journal on Computing, 2*: 4-32 (Winter 1990b).

Glover, F. *Tabu Search Fundamentals and Uses*. Working Paper. University of Colorado, Boulder CO, (April 1995).

Glover, F. and M. Laguna. *Tabu Search.* Boston: Kluwer Academic Publishers, (1997).

Glover, F., J.P. Kelly, and M. Laguna. "New advances and applications of combining simulation and optimization," *Proceedings of the 1996 Winter Simulation Conference*. Ed. J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain. Coronado CA, (8-11 December 1996).

Golden, B. L. and A. A. Assad, Eds. *Vehicle Routing: Methods and Studies.* Amsterdam: North Holland Press, (1988).

Hall, R. W. and J. G. Partyka. "On the Road to Efficiency," *OR/MS Today*. June 1997.

Kassou, I. and Pecuchet, J. "Use of simulation within a general framework of optimization in job-shop sheduling," *Proceedings of CISS - First Joint Conference of International Simulation Societies*. SCS: San Diego (1994).

Kervahut, T., B. Garcia, and J. Rousseau. "The Vehicle Routing Problem with Time Windows, Part 1: Tabu Search," *INFORMS Journal on Computing*. *8*: 158-164 (Spring 1996).

Klaf, A. A. *Trigonometry Refresher for Technical Men.* New York: McGraw Hill Book Co., Inc. (1946).

Laporte, Gilbert. "The Traveling Salesman Problem: An overview of exact and

approximate algorithms," *European Journal of Operational Research, 59*: 231-247 (1992).

Laporte, Gilbert. "The Vehicle Routing Problem: An overview of exact and approximate algorithms," *European Journal of Operational Research, 59*: 345-358 (1992).

Reinelt, G. "TSPLIB - A TSP Library", *ORSA Journal on Computing, 3*: 376-384, (1991).

Rego, Cesar and Catherine Roucairol. "A Parallel Tabu Search Algorithm Using Ejection Chains for the Vehicle Routing Problem," *Meta-Heuristics: Theory and Applications.* Ed. I. H. Osman and J. P. Kelly. Boston: Kluwer Academic Publishers, (1996).

Rochat, Yves and Eric D. Taillard. "Probabilistic Diversification and Intensification in Local Search for Vehicle Routing," *Journal of Heuristics, 1*: 147-167 (1995).

Savelsbergh, M. W. P. *Computer Aided Routing*. CWI Tract 75, CWI Amsterdam, (1992).

Sisson, Mark R. *Applying Tabu Heuristic to Wind Influenced, Minimum Risk and Maximum Expected Coverage Routes.* AFIT/GOR/ENS/97M. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, (February 1997).

Solomon, M.M. "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations Research, 35*: 254-265, (1987).

Woodruff, D. and E. Zemel. "Hashing Vectors for Tabu Search," *Annals of Operations Research*, 41: 123-137, (1993).

Vita

Joel L. Ryan was born in Tuscaloosa, Alabama, on 4 March 1970, the son of Col. Robert and Nancy Ryan while then Capt. Robert Ryan was enrolled in an AFIT/CI program at the University of Alabama. After graduating from Samuel Clemens High School in Schertz, Texas, he enrolled in Texas Tech University in 1988. Buoyed by a nomination from the Air Force R.O.T.C. detachment at Texas Tech, he transferred to the Air Force Academy the following year. Upon graduation in 1993, he accepted his first assignment to HQ Air Force Recruiting Service at Randolph AFB, Texas. He entered the Graduate School of Engineering, Air Force Institute of Technology, in August of 1996. After graduation, Capt. Ryan is slated for an assignment to the Studies and Analysis Squadron of HQ Air Combat Command, Langley AFB, Virginia.

Permanent Address: 912 Antler Dr.
Schertz, Texas 78154

Appendix A: tabuMod

The "tabuMod" library contains data structures and procedures useful to building any tabu search. The node and tour data types, as well as the VRP penalty record and coordinates array, are all created. The procedures include simple output, swap and insert moves, and the steps taken when a cycle is found or not found, as well as penalty, wait time, and vehicle count calculations.

```
DEFINITION MODULE tabuMod;

FROM IOMod IMPORT StreamObj, ALL FileUseType, ReadKey;
FROM hashMod IMPORT hashRecord;

TYPE

        nodeType = RECORD
                {input data}
                id,                         {node number}
                ea,                         {early time window (service start}
                la,                         {late time window (service start)}
                qty,                        {node demand or zero}
                type,                       {node type: 1-customer,2-vehicle}

                {schedule variables}
                arr,                        {arrival time}
                dep,                        {departure time}
                wait,                       {wait time}
                load      : INTEGER;        {total load on vehicle at visit}
        END RECORD; {nodeType}

        coordType = RECORD                  {node coordinates}
                x,y : REAL;
        END RECORD; {coordType}

        vrpPenType = RECORD   {penalties}
                tw, ld    : INTEGER;
        END RECORD;

        tourType = ARRAY INTEGER OF nodeType;

        coordArrType = ARRAY INTEGER OF coordType;

        arrRealType = ARRAY INTEGER OF REAL;
        arrInt2dimType = ARRAY INTEGER, INTEGER OF INTEGER;
        arrIntType = ARRAY INTEGER OF INTEGER;
        arrReal2dimType = ARRAY INTEGER, INTEGER OF REAL;
```

```
{Output the required info through various methods}

{Outputs the tour info to the screen}
PROCEDURE tourToScreen(IN nc, nv, numnodes :INTEGER;
                        IN coord : coordArrType;
                        IN tour : tourType);

{Like tourToScreen, but to the out file}
PROCEDURE tourToFile(IN where : STRING;
                        IN outstrm :StreamObj;
                        IN coord : coordArrType;
                        IN tour : tourType;
                        IN nc, nv, numnodes,
                        tourLen :INTEGER;
                        IN m : arrIntType);

{Outputs time matrix to out file}
PROCEDURE timeToFile(IN where : STRING;
                        IN outstrm :StreamObj;
                        IN time : arrInt2dimType;
                        IN numnodes : INTEGER);

{Like tourToFile, but much more clear and takes into account if load info
is viable}
PROCEDURE twLoadToFile(IN where : STRING;
                   IN outstrm :StreamObj;
                   IN tour : tourType;
                   IN nc, numnodes,
                   tourLen :INTEGER;
                   IN factor : REAL;
                   IN load : BOOLEAN);

{Similar to tourToFile, puts coordinates to file so you can scatter plot results}
PROCEDURE LatLongToFile(IN where : STRING;
                   IN outstrm :StreamObj;
                   IN tour : tourType;
                   IN nc, numnodes :INTEGER;
                   IN coord : coordArrType);

{Puts out only the node id, type, and order info of the tour to the file}
PROCEDURE qcktourFile(IN outstrm :StreamObj;
                   IN tour : tourType;
                   IN numnodes :INTEGER);

{*** END OF OUTPUT PROCEDURES ***}

{swap 2 integer variables}
PROCEDURE SwapInt(INOUT a, b : INTEGER);

{swap 2 nodeType variables}
PROCEDURE SwapNode(INOUT a, b : nodeType);

{Computes the tour schedule parameters for computing the schedule parameters
 for a tour. It returns the total tour length}
PROCEDURE tourSched(IN is,          {first customer node in tour}
```

```
                    nc, numnodes : INTEGER;
                    INOUT tour : tourType;
                    IN time : arrInt2dimType;
                    OUT tourLen : INTEGER;
                    IN outstrm : StreamObj);

{Find the number of vehicles being used in the current tour,
 by counting the vehicle to demand transitions}
PROCEDURE countVeh(IN numnodes : INTEGER;
                IN tour : tourType;
                OUT nvu : INTEGER);


{Calculate TW and LOAD penalties, store in tourPen record}
PROCEDURE compPens(IN numnodes : INTEGER;
                IN tour : tourType;
                IN capacity : INTEGER;
                INOUT tourPen : vrpPenType);

{given the TW and LOAD penalties, this procedure personalizes the penalties to the mTSPTW; Computes
cost of tour as tour length + penalty for infeasibilities}
PROCEDURE tsptwPen (IN numnodes, tourLen : INTEGER;  {length of curr tour}
                    IN tour : tourType;              {current tour}
                    IN tourPen : vrpPenType;  {record of TW & LD pens}
                    IN TWPEN : REAL;                 {mult factor for TW pen}
                    OUT totPenalty,           {total Penalty (TW here}
                        tourCost,                    {tourLen + TW cost}
                        penTrav,                     {tourCost - totWait}
                        tvl : INTEGER );             {travel time}

{Compute the sum of the waiting time in a given tour}
PROCEDURE sumWait (IN numnodes : INTEGER;
                IN tour : tourType;
                OUT sumwait : INTEGER);


{Updates the search parameters if the incumbent tour is not found in the
 hashing structure}
PROCEDURE nocycle (IN DECREASE    : REAL;          {RTS decrease tabuLen parameter}
                IN minTL      : INTEGER;
                IN mavg       : REAL;          {moving average of cycle length}
                INOUT ssltlc,    {steps since last tabu length change}
                    tabuLen : INTEGER;     {tabu length}
                IN outstrm    : StreamObj;     {output file}
                IN cycleprint : BOOLEAN);

{Updates the search parameters if the incumbent tour is found in the
 hashing structure}
PROCEDURE cycle (INOUT matchptr     : hashRecord;     {current tour's hash info}
               IN INCREASE  : REAL;          {RTS increase tabuLen parameter}
                IN maxTL,
                    CYMAX,                    {max cyleLength used to alter mavg}
                    k        : INTEGER;      {current iteration number}
               INOUT mavg    : REAL;          {cycle length moving average}
               INOUT  ssltlc,    {steps since last tabu length change}
```

```
                    tabuLen : INTEGER;          {tabu length}
                IN outstrm      : StreamObj;    {output file}
                IN cycleprint : BOOLEAN);

{Computes the incremental change in the value of the travel time from the incumbent
 tour to the proposed neighbor tour, and computes the neighbor schedule parameters
 preparing for computation of penalty terms (see compPens)}
PROCEDURE moveValTT (IN  i,                     {position of customer to be moved}
                    d,                          {depth of the insertion}
                    numnodes : INTEGER;
                IN tour         : tourType;     {current tour}
                INOUT nbrtour : tourType;       {neighbour, temporary tour}
                IN time : arrInt2dimType;
                OUT moveVal : INTEGER);
{adjusts the current tour for the defined insert move}
PROCEDURE insert (IN chI, chD : INTEGER; {origin and recipient of insert move}
                INOUT tour      : tourType);    {current tour}

END MODULE.


IMPLEMENTATION MODULE tabuMod;

FROM IOMod IMPORT StreamObj, ALL FileUseType, ReadKey;
FROM OSMod IMPORT SystemTime;
FROM hashMod IMPORT hashRecord;


{Output the required info in various forms}

PROCEDURE tourToScreen(IN nc, nv, numnodes :INTEGER;
                    IN coord : coordArrType;
                    IN tour : tourType);
VAR
    i : INTEGER;
BEGIN

    OUTPUT("Node information follows:");
    OUTPUT("  # nodes = ",numnodes+1);
    FOR i := 0 TO numnodes
      IF i = 0
        OUTPUT("DEPOT ",coord[i].x," ",coord[i].y);

      ELSIF i > nc
        OUTPUT("VEHICLE");
        OUTPUT("INPUT ",tour[i].id," ",tour[i].ea," ",tour[i].la
            ," ",tour[i].qty," ",tour[i].type," SCHED ",
            tour[i].arr," ",tour[i].dep," ",tour[i].wait,
            " ",tour[i].load);

      ELSE
        OUTPUT("NODE ",coord[i].x," ",coord[i].y);
        OUTPUT("INPUT ",tour[i].id," ",tour[i].ea," ",tour[i].la
            ," ",tour[i].qty," ",tour[i].type," SCHED ",
            tour[i].arr," ",tour[i].dep," ",tour[i].wait,
```

```
                " ",tour[i].load);
        END IF;
      END FOR;

END PROCEDURE; {tourToScreen}

PROCEDURE tourToFile(IN where : STRING;
                     IN outstrm :StreamObj;
                     IN coord : coordArrType;
                     IN tour : tourType;
                     IN nc, nv, numnodes,
                     tourLen :INTEGER;
                     IN m : arrIntType);
CONST
{      id  x   y eArr lArr | Arr  Dep Wait | Qty Load   Mid}
format="***<***<***< ***< ***<   ***<***<***<  **< ***<   ***<";
VAR
      i : INTEGER;
      name, str : STRING;
BEGIN
      ASK outstrm WriteString(where);
      ASK outstrm WriteLn;
      ASK outstrm WriteString("Tour Length: ");
      ASK outstrm WriteInt(tourLen,4);
      ASK outstrm WriteLn;
      ASK outstrm WriteString("Node information follows:");
      ASK outstrm WriteLn;
      ASK outstrm WriteString("TYPE  ID   x   y | eArr lArr | Arr Dep");
      ASK outstrm WriteString(" Wait | Qty Load | Mid");
      ASK outstrm WriteLn;

      FOR i := 0 TO numnodes
        IF (tour[i].id = 0) OR (tour[i].id = numnodes)
          name := "DEPOT ";
        ELSIF tour[i].type = 2
          name := "VHCL  ";
        ELSIF tour[i].type = 1
          name := "NODE  ";
        END IF;

        IF tour[i].type = 1
      str := SPRINT (tour[i].id, coord[i].x, coord[i].y,
                     tour[i].ea, tour[i].la, tour[i].arr, tour[i].dep,
                     tour[i].wait, tour[i].qty, tour[i].load, m[i])
              WITH format;
        ELSE
      str := SPRINT (tour[i].id, coord[0].x, coord[0].y,
                     tour[i].ea, tour[i].la, tour[i].arr, tour[i].dep,
                     tour[i].wait, tour[i].qty, tour[i].load)
              WITH format;
        END IF;

        ASK outstrm WriteString(name);
        ASK outstrm WriteString(str);
        ASK outstrm WriteLn;
```

```
        END FOR;

        ASK outstrm WriteLn;

END PROCEDURE; {tourToFile}

PROCEDURE timeToFile(IN where : STRING;
                        IN outstrm :StreamObj;
                        IN time : arrInt2dimType;
                        IN numnodes : INTEGER);
VAR
        i,j : INTEGER;
BEGIN
        ASK outstrm WriteString(where);
        ASK outstrm WriteLn;
        ASK outstrm WriteString("    ");

        FOR i := 0 TO numnodes
                ASK outstrm WriteInt(i, 6);
        END FOR;
        ASK outstrm WriteLn;
        FOR i := 0 TO numnodes
                ASK outstrm WriteInt(i,6);
                FOR j := 0 TO numnodes
                        ASK outstrm WriteInt(time[i][j], 6)
                END FOR;
                ASK outstrm WriteLn;
        END FOR;

        ASK outstrm WriteLn;

END PROCEDURE; {timeToFile}


PROCEDURE twLoadToFile(IN where : STRING;
                        IN outstrm :StreamObj;
                        IN tour : tourType;
                        IN nc, numnodes,
                        tourLen :INTEGER;
                        IN factor : REAL;
                        IN load : BOOLEAN);
CONST
        {     id  eArr   lArr  l Arr    Dep     Wait  l Qty Load}
        format1="***< ****.*< ****.*<  *****.*< *****.*< ***.*<   **< ***<";
        format2="***< ****.*< ****.*<  *****.*< *****.*< ***.*<";
VAR
        i : INTEGER;
        name, str : STRING;
BEGIN
ASK outstrm WriteString(where);
ASK outstrm WriteLn;
ASK outstrm WriteString("Tour Length: ");
ASK outstrm WriteInt(tourLen,4);
ASK outstrm WriteLn;
ASK outstrm WriteString("Node information follows:");
```

```
ASK outstrm WriteLn;
IF load
  ASK outstrm WriteString("TYPE ID   eArr   lArr   | Arr     Dep     Wait");
  ASK outstrm WriteString("| Qty Load");
ELSE
  ASK outstrm WriteString("TYPE ID   eArr   lArr   | Arr     Dep     Wait");
END IF;
ASK outstrm WriteLn;

FOR i := 0 TO numnodes
   IF ((tour[i].id = 0) OR (tour[i].id = numnodes))
      AND (tour[i].type = 2)
     name := "DEPOT ";
   ELSIF tour[i].type = 2
     name := "VHCL  ";
   ELSIF tour[i].type = 1
     name := "NODE  ";
   END IF;

   IF load = TRUE
     str := SPRINT(tour[i].id, FLOAT(tour[i].ea) / factor,
          FLOAT(tour[i].la) / factor, FLOAT(tour[i].arr) / factor,
          FLOAT(tour[i].dep) / factor, FLOAT(tour[i].wait) / factor,
          tour[i].qty, tour[i].load )
       WITH format1;
   ELSE
     str := SPRINT(tour[i].id, FLOAT(tour[i].ea) / factor,
           FLOAT(tour[i].la) / factor, FLOAT(tour[i].arr) / factor,
           FLOAT(tour[i].dep) / factor, FLOAT(tour[i].wait) / factor)
       WITH format2;
   END IF;

   ASK outstrm WriteString(name);
   ASK outstrm WriteString(str);
   ASK outstrm WriteLn;
END FOR;

ASK outstrm WriteLn;

END PROCEDURE; {twLoadToFile}

{Similar to tourToFile, puts coordinates to file so you can scatter plot results}
PROCEDURE LatLongToFile(IN where : STRING;
                        IN outstrm :StreamObj;
                        IN tour : tourType;
                        IN nc, numnodes :INTEGER;
                        IN coord : coordArrType);
CONST
        {    id   x     y}
        format1="***< ***.****< ***.****<";
VAR
        i : INTEGER;
        name, str : STRING;
BEGIN
ASK outstrm WriteString(where);
```

```
ASK outstrm WriteLn;

ASK outstrm WriteString("TYPE  ID  coordX coordY");

FOR i := 0 TO numnodes
  IF ((tour[i].id = 0) OR (tour[i].id = numnodes))
    AND (tour[i].type = 2)
   name := "DEPOT ";
  ELSIF tour[i].type = 2
   name := "VHCL  ";
  ELSIF tour[i].type = 1
   name := "NODE  ";
  END IF;

  str := SPRINT(tour[i].id, coord[tour[i].id].x, coord[tour[i].id].y)
      WITH format1;

  ASK outstrm WriteString(name);
  ASK outstrm WriteString(str);
  ASK outstrm WriteLn;
END FOR;

ASK outstrm WriteLn;

END PROCEDURE; {LatLongToFile}

{just sends a "quick" tour permutation to the out file}
PROCEDURE qcktourFile(IN outstrm :StreamObj;
                      IN tour : tourType;
                      IN numnodes :INTEGER);
        CONST
        VAR
                i : INTEGER;
                type : STRING;
                str : STRING;
        BEGIN

        FOR i := 0 TO numnodes
         IF (i = 0) AND (tour[i].type = 2)
           type := "D";
         ELSIF (i=numnodes) AND (tour[i].type = 2)
           type :=  "D";
         ELSIF tour[i].type = 2
           type := "V";
         ELSE
           type := "C";
         END IF;
         str := type + INTTOSTR(tour[i].id) + " ";
         ASK outstrm WriteString(str);
        END FOR;

        ASK outstrm WriteLn;

END PROCEDURE; {qcktourFile}
```

```
{*** END OF OUTPUT PROCEDURES ***}


{swap 2 integer variables}
PROCEDURE SwapInt(INOUT a, b : INTEGER);
VAR
        temp : INTEGER;
BEGIN
        temp := a;
        a := b;
        b := temp;
END PROCEDURE; {Swap}

{swap 2 nodeType variables}
PROCEDURE SwapNode(INOUT a, b : nodeType);
VAR
        temp : nodeType;
BEGIN
        temp := a;
        a := b;
        b := temp;
END PROCEDURE; {Swap}

{Computes the tour schedule parameters for computing the schedule parameters
 for a tour.  It returns the total tour length}
PROCEDURE tourSched(IN is,              {first customer node in tour}
                        nc, numnodes : INTEGER;
                        INOUT tour : tourType;
                        IN time : arrInt2dimType;
                        OUT tourLen : INTEGER;
                        IN outstrm : StreamObj);
VAR
        i,k,continue, lastnode      : INTEGER;
BEGIN
        tourLen := 0;

        {Compute the tour length from depot to node is }
        i := 0;
        WHILE i < is-1
                tourLen := tourLen + time[tour[i].id][tour[i+1].id]
                                + tour[i+1].wait;
                i := i + 1;
        END WHILE;

        {update the schedule from is to the last node}
        FOR i := is-1 TO numnodes-1
        {load update}
                IF tour[i+1].type = 2
                        tour[i+1].load := tour[i+1].qty;
                ELSE
                        tour[i+1].load := tour[i].load + tour[i+1].qty;
                END IF;

        {find arrival times}
```

```
            tour[i+1].arr := tour[i].dep +
                                time[tour[i].id][tour[i+1].id];

        {find departure and wait times}
            IF tour[i+1].type = 2
                tour[i+1].dep := tour[i+1].ea;
                tour[i+1].wait := 0;
            ELSE
                tour[i+1].dep := MAXOF(tour[i+1].ea, tour[i+1].arr);
                tour[i+1].wait := tour[i+1].dep - tour[i+1].arr;
            END IF;

        {tourLen update}
            tourLen := tourLen + time[tour[i].id][tour[i+1].id] + tour[i+1].wait;

        END FOR;

END PROCEDURE; {tourSched}

{Find the number of vehicles being used in the current tour,
 by counting the vehicle to demand transitions}
PROCEDURE countVeh(IN numnodes : INTEGER;
                   IN tour : tourType;
                   OUT nvu : INTEGER);
VAR
        i : INTEGER;
BEGIN
        nvu := 0;

        FOR i := 0 TO numnodes-1
            IF (tour[i].type = 2) AND (tour[i+1].type = 1)

                nvu := nvu + 1;

            END IF;
        END FOR;

END PROCEDURE; {countVeh}

{Computes the exact vehicle OVERLOAD and TIME WINDOW penalties}
PROCEDURE compPens(IN numnodes : INTEGER;
                   IN tour : tourType;
                   IN capacity : INTEGER;
                   INOUT tourPen : vrpPenType);
VAR
        i, infeastw, infeasld : INTEGER;
BEGIN
        infeastw := 0;
        infeasld := 0;

        FOR i := 1 TO numnodes
            infeastw := infeastw + MAXOF(0, tour[i].arr - tour[i].la);
            IF (tour[i].type = 2) AND (capacity > 0)
              infeasld := infeasld + MAXOF(0, tour[i].load - capacity);
```

```
            END IF;
        END FOR;
        tourPen.tw := infeastw;
        tourPen.ld := infeasld;

END PROCEDURE; {compPens}

{given the TW and LOAD penalties, this procedure personalizes the penalties to the mTSPTW; Computes
cost of tour as tour length + penalty for infeasibilities}
PROCEDURE tsptwPen (IN numnodes, tourLen : INTEGER; {length of curr tour}
                    IN tour : tourType;            {current tour}
                    IN tourPen : vrpPenType; {record of TW & LD pens}
                    IN TWPEN : REAL;               {mult factor for TW pen}
                    OUT totPenalty,          {total Penalty (TW here}
                        tourCost,                  {tourLen + TW cost}
                        penTrav,                   {tourCost - totWait}
                        tvl : INTEGER );           {travel time}
VAR
        i, totWait, twCost : INTEGER;

BEGIN
        {compute infeasibilities}

        totPenalty := tourPen.tw;

        {compute tour infeasibility costs}
        twCost := TRUNC( TWPEN * FLOAT(totPenalty) );

        {compute tour characteristic values}
        tourCost := tourLen + twCost;

        sumWait(numnodes, tour, totWait);

        penTrav := tourCost - totWait;
        tvl := penTrav - twCost;

END PROCEDURE; {tsptwPen}

{Compute the sum of the waiting time in a given tour}
PROCEDURE sumWait (IN numnodes : INTEGER;
                   IN tour : tourType;
                   OUT sumwait : INTEGER);
VAR
        i : INTEGER;
BEGIN
        sumwait := 0;

        FOR i := 0 TO numnodes
                sumwait := sumwait + tour[i].wait;
        END FOR;

END PROCEDURE; {sumWait}

{Updates the search parameters if the incumbent tour is not found in the
 hashing structure}
```

```
PROCEDURE nocycle (IN DECREASE      : REAL;           {RTS decrease tabuLen parameter}
                   IN minTL        : INTEGER;
                   IN mavg         : REAL;           {cycle length moving average}
                   INOUT ssltlc,     {steps since last tabu length change}
                         tabuLen : INTEGER;       {tabu length }
                   IN outstrm      : StreamObj;       {output file}
                   IN cycleprint : BOOLEAN);
BEGIN
        ssltlc := ssltlc + 1;

        IF FLOAT(ssltlc) > mavg
{NOTE:  tabuLen always > 5. If tabuLen were < 5, it would never increase w/ INCREASE = 1.2}
                {Adjust tabuLen}
                tabuLen := MAXOF( TRUNC(FLOAT(tabuLen)*DECREASE), minTL);
                ssltlc := 0;
        END IF;

IF cycleprint
        ASK outstrm WriteString("The tour was not found in the hash structure");
        ASK outstrm WriteString("   The current mavg: ");
        ASK outstrm WriteReal(mavg, 7, 1);
        ASK outstrm WriteLn;
        ASK outstrm WriteString("   Steps since last tabuLen change: ");
        ASK outstrm WriteInt(ssltlc,6);
        ASK outstrm WriteString("   Current tabuLen:");
        ASK outstrm WriteInt(tabuLen,6);
        ASK outstrm WriteLn;
END IF;

END PROCEDURE; {nocycle}

{Updates the search parameters if the incumbent tour is found in the
 hashing structure}
PROCEDURE cycle (INOUT matchptr       : hashRecord;      {current tour's hash info}
                 IN INCREASE   : REAL;            {RTS increase tabuLen parameter}
                  IN maxTL,
                        CYMAX,                            {max cyleLength used to alter mavg}
                        k          : INTEGER;       {current iteration number}
                 INOUT mavg     : REAL;           {cycle length moving average}
                 INOUT  ssltlc,     {steps since last tabu length change}
                        tabuLen : INTEGER;       {tabu length}
                  IN outstrm      : StreamObj;       {output file}
                  IN cycleprint : BOOLEAN);
VAR
        cycleLength  : INTEGER;  {cycle length for the found tour}
BEGIN
        ssltlc := ssltlc + 1;

        cycleLength := k - matchptr.lastiter;

        {update when hash record last visited}
        matchptr.lastiter := k;

        IF cycleLength < CYMAX
                mavg := 0.1 * FLOAT(cycleLength) + 0.9 * mavg;
```

```
            tabuLen := MINOF(maxTL, TRUNC( FLOAT(tabuLen)*INCREASE ) );
        END IF;

IF cycleprint
        ASK outstrm WriteString("The tour was not found in the hash structure");
        ASK outstrm WriteString("   The current mavg: ");
        ASK outstrm WriteReal(mavg,7,1);
        ASK outstrm WriteLn;
        ASK outstrm WriteString("   Steps since last tabuLen change: ");
        ASK outstrm WriteInt(ssltlc,6);
        ASK outstrm WriteString("   Current tabuLen:");
        ASK outstrm WriteInt(tabuLen,6);
        ASK outstrm WriteLn;
END IF;

END PROCEDURE; {cycle}

{Computes the incremental change in the value of the travel time from the incumbent
 tour to the proposed neighbor tour, and computes the neighbor schedule parameters
 preparing for computation of penalty terms (see compPens)}
PROCEDURE moveValTT (IN  i,                    {position of customer to be moved}
                         d,                    {depth of the insertion}
                         numnodes : INTEGER;
                     IN tour      : tourType;  {current tour}
                     INOUT nbrtour : tourType; {neighbor, temporary tour}
                     IN time : arrInt2dimType;
                     OUT moveVal : INTEGER);
VAR
        is,                     {predecessor of the moving node in its old spot}
        j,                      {predecessor of the moving node in its old spot}
        delin,                  {incremental tour travel time, entering arcs}
        delout,                 {incremental tour travel time, leaving arcs}
        iend,                   {index to the end of "within" area of insertion}

        nodelf,                 {tour index of node at left}
        nodert    : INTEGER;    {tour index of node at right}
BEGIN
        delin := 0; delout := 0;

        IF d > 0
            j := i + d;
            is := i + d - 1;
            iend := i + d + 1;
        ELSE
            j := i + d -1;
            is := i + d;
            iend := i + d + 3;
        END IF;

        nodelf := is-1;
        nodert := nodelf + 1;

        {updates the schedule from node is to appropriate vehicle or terminal depot}
        WHILE (nodelf < iend) OR (nbrtour[nodelf].type <> 2)
```

```
    {update arrival}
    nbrtour[nodert].arr := nbrtour[nodelf].dep +
                           time[nbrtour[nodelf].id][nbrtour[nodert].id];
    {update dep and wait times}
    IF nbrtour[nodert].type = 2
     nbrtour[nodert].dep := nbrtour[nodert].ea;
     nbrtour[nodert].wait := 0;
     nbrtour[nodert].load := 0;
    ELSE
     nbrtour[nodert].load := nbrtour[nodelf].load + nbrtour[nodert].qty;
     nbrtour[nodert].dep := MAXOF(nbrtour[nodert].ea, nbrtour[nodert].arr);
     nbrtour[nodert].wait := nbrtour[nodert].dep - nbrtour[nodert].arr;
    END IF;

    nodelf := nodelf + 1;
    nodert := nodert + 1;
   END WHILE;

   {Relative to the incumbent tour (tour) and working tour (nbrtour),
    compute the change in travel time}

   delout := time[tour[i-1].id][tour[i].id] + time[tour[i].id][tour[i+1].id]
             + time[tour[j].id][tour[j+1].id];

   delin := time[tour[i-1].id][tour[i+1].id] + time[tour[j].id][tour[i].id]
             + time[tour[i].id][tour[j+1].id];

   moveVal := delin - delout;

END PROCEDURE; {moveValTT}

{adjusts the current tour for the defined insert move}
PROCEDURE insert (IN chI, chD : INTEGER; {origin and recipient of insert move}
                  INOUT tour    : tourType);       {current tour}
VAR
   i, j  : INTEGER;
BEGIN

   IF chD > 0
         FOR j := 0 TO chD-1
               SwapNode(tour[chI+j], tour[chI+j+1]);
         END FOR;
   ELSE
         FOR j := 0 DOWNTO chD+1
               SwapNode(tour[chI+j], tour[chI+j-1]);
         END FOR;
   END IF;

END PROCEDURE; {insert}

END MODULE.
```

Appendix B: tsptwMod

The library "tsptwMod" contains the objects, methods, and procedures related to the mTSPTW. Objects include a timeMatrixObj meant for reading in the problem data and calculating the time matrix. The object startTourObj reorders the initial tour by the time window medians and initializes the parameters associated with finding a best tour. The object reacTabuObj contains one method, the reactive tabu search created by Carlton (1995). The implementation module follows.

```
IMPLEMENTATION MODULE tsptwMod;

FROM IOMod IMPORT StreamObj, ALL FileUseType;
FROM OSMod IMPORT SystemTime;
FROM MathMod IMPORT SQRT;
        {VRP data types}
FROM tabuMod IMPORT arrInt2dimType;
FROM tabuMod IMPORT arrIntType;
FROM tabuMod IMPORT arrRealType;
FROM tabuMod IMPORT coordType;
FROM tabuMod IMPORT coordArrType;
FROM tabuMod IMPORT nodeType;
FROM tabuMod IMPORT tourType;
FROM tabuMod IMPORT vrpPenType;
        {output stuff}
FROM tabuMod IMPORT qcktourFile;
FROM tabuMod IMPORT twLoadToFile;
        {Move/order/search stuff}
FROM tabuMod IMPORT SwapInt;
FROM tabuMod IMPORT SwapNode;
FROM tabuMod IMPORT moveValTT;  {Travel time version}
FROM tabuMod IMPORT insert;
        {schedule, penalty, and hash stuff}
FROM tabuMod IMPORT tourSched;
FROM tabuMod IMPORT compPens;
FROM tabuMod IMPORT tsptwPen;  {TSPTW version}
        {best solution tracker}
FROM bestSolnMod IMPORT twBestTT; {best travel time, lowest number of vehicles}
        {hashing stuff}
FROM hashMod IMPORT hashListObj;
FROM hashMod IMPORT hashRecord;
FROM hashMod IMPORT hashTblType;
FROM hashMod IMPORT randWtWZ;
FROM hashMod IMPORT tourHVwz;
FROM hashMod IMPORT lookfor;
        {reaction stuff}
```

```
FROM tabuMod IMPORT nocycle;
FROM tabuMod IMPORT cycle;

FROM tabuMod IMPORT countVeh;

OBJECT timeMatrixObj;
{Reads in the x,y coordinates and time window file and calculates the
 time between each node.  Does not assume the problem is symmetric, but
 makes it so}

        {read in the time matrix directly -- for mTSP problems}
        ASK METHOD readTime (IN instrm : StreamObj;
                             IN gamma, nv, maxtime : INTEGER;
                             OUT nc, numnodes : INTEGER;
                             IN factor : REAL;
                             OUT tour, bestTour, bfTour : tourType;
                             OUT time : arrInt2dimType);
        VAR
                i, j : INTEGER;
                node : nodeType;
        BEGIN

        ASK instrm ReadInt(nc);             {read in # customers}

        numnodes := nc + nv;

        NEW(tour, 0..numnodes);
        NEW(bfTour, 0..numnodes);
        NEW(bestTour, 0..numnodes);

        FOR i := 0 TO numnodes
                NEW(node);                  {instantiate each node}
                tour[i] := node;    {place each node in array}

                tour[i].id := i;            {set node id}

                IF (i = 0) OR (i > nc)              {set node types}
                        tour[i].type := 2;  {2=veh node}
                ELSE
                        tour[i].type := 1;  {1=cust node}
                END IF;

                tour[i].arr := 0;
                tour[i].dep := 0;
                tour[i].wait := 0;
                tour[i].load := 0;
                tour[i].ea := 0;
                tour[i].la := maxtime;

                bestTour[i] := CLONE(tour[i]);
                bfTour[i] := CLONE(tour[i]);
        END FOR;

        NEW(time, 0..numnodes, 0..numnodes);        {initialize time matrix}
```

```
{read in SYMMETRIC time matrix}
FOR i := 0 TO nc
        FOR j := i+1 TO nc
                ASK instrm ReadInt(time[i][j]);
                time[j][i] := time[i][j];
        END FOR;
END FOR;

{demand to vehicle & vehicle to demand travel same as demand to depot}
FOR i := 1 TO nc
        FOR j := nc+1 TO numnodes
                time[i][j] := time[0][i];
                time[j][i] := time[i][j];
        END FOR;
END FOR;

END METHOD; {readTime}

{Reads in the x,y coordinates for a simple symmetric TSP problem}
{AND calculates the time matrix}
ASK METHOD readTSP(IN instrm : StreamObj;
                        OUT nc, numnodes : INTEGER;
                        IN nv, maxtime : INTEGER;
                   OUT tour, bestTour, bfTour : tourType;
                   OUT time : arrInt2dimType);
VAR
        i, j, id        : INTEGER;
        xdiff, ydiff,
        xdiff2, ydiff2 : REAL;
        position : coordType;       {record to instantiate array of coord}
        coord    : coordArrType;
        node     : nodeType;        {record to instantiate array of nodes}
BEGIN
ASK instrm ReadInt(nc);             {read in # customers}

numnodes := nc + nv;        {# nodes in directed graph}

NEW(tour, 0..numnodes);
NEW(bfTour, 0..numnodes);
NEW(bestTour, 0..numnodes);


FOR i := 0 TO numnodes
        NEW(node);                  {instantiate each node}
        tour[i] := node;    {place each node in array}

        tour[i].id := i;            {set node id}

        IF (i = 0) OR (i > nc)              {set node types}
                tour[i].type := 2;  {2=veh node}
        ELSE
                tour[i].type := 1;  {1=cust node}
        END IF;

        tour[i].arr := 0;
```

```
        tour[i].dep := 0;
        tour[i].wait := 0;
        tour[i].load := 0;
        tour[i].ea := 0;
        tour[i].la := maxtime;

        bestTour[i] := CLONE(tour[i]);
        bfTour[i] := CLONE(tour[i]);
END FOR;

NEW(time, 0..numnodes, 0..numnodes);      {initialize time matrix}

NEW(coord, 1..nc);        {initialize array of positions}

FOR i := 1 TO nc
        NEW(position);              {instantiate each record}
        coord[i] := position;       {place record in array}
END FOR;

{read in customer nodes}
FOR i := 1 TO nc
        ASK instrm ReadInt(id);
        ASK instrm ReadReal(coord[i].x);
        ASK instrm ReadReal(coord[i].y);
END FOR;

{initialize vehicle nodes and terminal depot}
FOR i := nc+1 TO numnodes
        tour[i] := CLONE(tour[0]);
        tour[i].id := i;
END FOR;

{Find integer euclidean dist}
{depot and demand nodes}
FOR i := 1 TO nc
        FOR j := i+1 TO nc
                xdiff := coord[i].x - coord[j].x;
                ydiff := coord[i].y - coord[j].y;
                xdiff2 := xdiff*xdiff;
                ydiff2 := ydiff*ydiff;
                time[i][j] := TRUNC(SQRT(xdiff2 + ydiff2));
                time[j][i] := time[i][j];
        END FOR;
END FOR;

{depot to demand & demand to depot travel all equal 0 !!}
FOR i := 1 TO nc
        time[0][i] := 0;
        time[i][numnodes] := 0;
END FOR;

END METHOD; {readTSP}

{Reads in the x,y coordinates and time window file and calculates the
time between each node. }
```

```
{reads in a dataset of Solomon's style}
ASK METHOD readTSPTW(IN instrm : StreamObj;
                     OUT nc, numnodes : INTEGER;
                     IN factor : REAL;
                     IN nv : INTEGER;
                     OUT coord : coordArrType;
                     OUT tour : tourType;
                     OUT s : arrIntType );
VAR
    i, id,
    qty : INTEGER;          {quantity demanded at each node}
    servtime,               {service time at node}
    late,                   {late start to TW}
    early    : REAL;        {early start to TW}
    position : coordType;   {record to instantiate array of coord}
    node     : nodeType;    {record to instantiate array of nodes}
BEGIN
    ASK instrm ReadInt(nc);             {read in # customers}

    numnodes := nc + nv;    {# nodes in directed graph}

    NEW(tour, 0..numnodes); {initialize array of nodes}
                            {node 0=depot, nc cust node}
                            {nodes > nc are vehicle}
                            {nv-1 vehicle nodes, 0 is 1st vehicle}
                            {numnodes = terminal depot}
    FOR i := 0 TO numnodes
        NEW(node);                      {instantiate each node}
        tour[i] := node;    {place each node in array}

        tour[i].id := i;                {set node id}

        IF (i = 0) OR (i > nc)                  {set node types}
            tour[i].type := 2;  {2=veh node}
        ELSE
            tour[i].type := 1;  {1=cust node}
        END IF;
    END FOR;

    NEW(coord, 0..nc);      {initialize array of positions}
    NEW(s, 0..nc);          {initialize service time array}

    FOR i := 0 TO nc
        NEW(position);              {instantiate each record}
        coord[i] := position;       {place record in array}
    END FOR;

    {read in depot node}
    ASK instrm ReadReal(coord[0].x);
    ASK instrm ReadReal(coord[0].y);
    ASK instrm ReadInt(qty);
    ASK instrm ReadReal(early);
    ASK instrm ReadReal(late);
    ASK instrm ReadReal(servtime);
```

```
        s[0] := TRUNC(factor * servtime);
        tour[0].ea := TRUNC(factor*early);   {use Int times}
        tour[0].la := TRUNC(factor*late);

        {read in customer nodes}
        FOR i := 1 TO nc
                ASK instrm ReadReal(coord[i].x);
                ASK instrm ReadReal(coord[i].y);
                ASK instrm ReadInt(tour[i].qty);
                ASK instrm ReadReal(early);
                ASK instrm ReadReal(late);
                ASK instrm ReadReal(servtime);

                tour[i].ea := TRUNC(factor*early);   {use Int times}
                tour[i].la := TRUNC(factor*late);
                s[i] := TRUNC(factor * servtime);
        END FOR;

        {initialize depot node}
        tour[0].type := 2;
        tour[0].arr := tour[0].ea;
        tour[0].dep := tour[0].ea;
        tour[0].wait := 0;
        tour[0].load := 0;

        {initialize vehicle nodes and terminal depot}
        FOR i := nc+1 TO numnodes
                tour[i] := CLONE(tour[0]);
                tour[i].id := i;
        END FOR;

END METHOD; {readTSPTW}

{Compute 2 dimensional time/distance matrix}
{Does not assume the problem is symmetric, but makes it so}
ASK METHOD timeMatrix(IN nc , numnodes, gamma : INTEGER;
                        IN factor : REAL;
                        IN tour : tourType;
                        IN coord : coordArrType;
                        OUT time : arrInt2dimType;
                        IN s : arrIntType );
VAR
        i, j,
        k : INTEGER;
        xdiff, ydiff,                    {differences for dist calc}
        xdiff2, ydiff2      : REAL;      {squared differences}
BEGIN
NEW(time, 0..numnodes, 0..numnodes);     {initialize time matrix}

{Find integer euclidean dist}
{depot and demand nodes}
FOR i := 0 TO nc-1
        FOR j := i+1 TO nc
                xdiff := coord[i].x - coord[j].x;
                ydiff := coord[i].y - coord[j].y;
```

```
                xdiff2 := xdiff*xdiff;
                ydiff2 := ydiff*ydiff;
                time[i][j] := TRUNC(factor * SQRT(xdiff2 + ydiff2));
                time[j][i] := time[i][j];
            END FOR;
        END FOR;

        {Ensure triangle inequality holds}
        FOR i := 0 TO nc-1
            FOR j := i+1 TO nc
                FOR k := 0 TO nc
                 IF (k <> i) AND (k <> j)
                  IF time[i][j] > time[i][k] + time[k][j]
                        time[i][j] := time[i][k] + time [k][j];
                        time[j][i] := time[i][j];
                  END IF;
                 END IF;
                END FOR;
            END FOR;
        END FOR;

        {demand to vehicle & vehicle to demand travel same as demand to depot}
        FOR i := 1 TO nc
            FOR j := nc+1 TO numnodes
                time[i][j] := time[0][i];
                time[j][i] := time[i][j];
            END FOR;
        END FOR;

        {Add service time for all demand/demand and demand/vehicle arcs}
        FOR i := 0 TO nc
            FOR j := 0 TO numnodes
                IF i <> j
                        time[i][j] := time[i][j] + s[i];
                END IF;
            END FOR;
        END FOR;

        {Add vehicle usage penalty "gamma" to all vehicle to vehicle arcs}
        FOR i := nc+1 TO numnodes-1
            time[0][i] := time[0][i] + gamma;
            FOR j := i+1 TO numnodes
                time[i][j] := time[i][j] + gamma;
                time[j][i] := time[i][j];
            END FOR;
        END FOR;

        END METHOD; {timeMatrix}

END OBJECT; {timeMatrixObj}


OBJECT startTourObj;
{Kicks off the clock, Computes an initial schedule and initial tour cost.
        Tour Cost= Travel time + Waiting Time + Penalty Term
```

Then computes the initial hashing values: f(T) and thv(T)}

```
{A. Produces a tour based on a sort of increasing avg time windows at
        each node. The customers are ordered by increasing avg time window
        value, and the nv vehicle nodes are appended to the end of the tour}

        ASK METHOD startTour (IN nv, nc : INTEGER;
                              IN time : arrInt2dimType;
                              INOUT tour : tourType;
                              OUT tourLen : INTEGER;
                              OUT totPenalty : INTEGER;
                              OUT tourhv,
                                startTime : INTEGER;
                              OUT m : arrIntType;
                              IN outstrm : StreamObj);   {**can remove m**}
        VAR
                i, j,
                numnodes : INTEGER;

        BEGIN

        numnodes := nc + nv;

        startTime := SystemTime();

        {1. compute the avg time window at each node "m[i]", exlude depot
        nodes.}

        NEW(m, 1..nc);
        FOR i := 1 TO nc
                m[i] := (tour[i].ea + tour[i].la) DIV 2 ;
        END FOR;

        {2. Bubble sort the initial tour based on avg TW time.  BUT, do not
        swap if the move would violate strong TW infeasibility}
        FOR i := 1 TO nc-1
                FOR j := nc DOWNTO i+1
                        IF (m[j-1] > m[j]) AND
                          (tour[j].ea + time[tour[j].id][tour[j-1].id]
                              <= tour[j-1].la)

                              SwapInt(m[j], m[j-1]);

                              SwapNode(tour[j], tour[j-1]);
                        END IF;
                END FOR;
        END FOR;
{B. Compute the initial schedule for the initial tour, and store the values in
the node structure. Also, returns the total tour length excluding any penalty
for infeasibility.  tour[0].ea = 0  here.}

        {Compute initial schedule, return tour's total travel + wait time}
        tourSched(1, nc, numnodes, tour, time, tourLen, outstrm);

        END METHOD; {startTour}
```

```
{C. Initialize "best" values and their times; Compute cost of initial tour as tour length + penalty for
infeasibilities}
        ASK METHOD startPenBest (IN numnodes, tvl,        {travel time of tour}
                                    tourLen : INTEGER;    {length of curr tour}
                                 IN tour : tourType;  {curr tour}
                                 IN TWPEN : REAL;             {mult factor for TW pen}
                                 OUT totPenalty,              {total Penalty (TW here)}
                                     penTrav,                 {tvl and twpen w/our wait}
                                     tourCost : INTEGER;      {tourLen + TW cost}
                                 OUT tourPen : vrpPenType;{tour penalty record}
                                 OUT bfiter,           {iter # of bf tour fd}
                                     bfCost,           {lowest feas cost found}
                                     bfTT,                    {best feas trav time}
                                     bfnv,
                                     bestiter,
                                     bestCost,                {lowest cost found}
                                     bestTT,
                                     bestnv,
                                  bfTime,              {Time best feas found}
                                  bestTime : INTEGER;         {Time best tour found}
                                 OUT bestTour,                {The best tour found}
                                   bfTour  : tourType); {best feas tour}
        VAR
                iter, nvu : INTEGER;
        BEGIN

        {initialize best FEASIBLE stuff}
        {make the initial best penalties really large}

        bfCost := 999999999;
        bfTT := 9999999;
        bfnv := 9999;
        bfTime := 0;
        bfiter := -1;

        NEW(bfTour, 0..numnodes);

        bestCost := 999999999;
        bestTT := 9999999;
        bestnv := 9999;
        bestTime := 0;
        bestiter := -1;

        NEW(bestTour, 0..numnodes);

        {compute infeasibilities and costs}
        {note: if totPenalty > 0, tour NOT feasible}
        NEW(tourPen);              {Tour Penalty record iniialized}

        compPens(numnodes, tour, 0, tourPen);
        tsptwPen(numnodes, tourLen, tour, tourPen, TWPEN, totPenalty, tourCost,
                penTrav, tvl);
```

```
        countVeh(numnodes, tour, nvu);
        twBestTT(numnodes, totPenalty, penTrav, tvl, nvu, 0, tour, bfCost, bfTT,
                    bfnv, bfiter, bestCost, bestTT, bestnv, bestiter, bfTour,
                    bestTour, bfTime, bestTime);

        END METHOD; {startPenBest}

END OBJECT; {startTourObj}

OBJECT reacTabuObj;

{Steps through ITER iterations of the reactive tabu search.}
ASK METHOD search (IN TWPEN, INCREASE, DECREASE : REAL;
                    IN HTSIZE, CYMAX, ZRANGE, DEPTH, minTL, maxTL, tabuLen,
                          iters, nc, numnodes : INTEGER;
                    IN outstrm, outstrm2  : StreamObj;
                    INOUT tourPen : vrpPenType;
                    IN time : arrInt2dimType;
                    IN stepprint, moveprint, cycleprint : BOOLEAN;
                    INOUT tourCost, penTrav, totPenalty, tvl,
                          bfCost, bfTT, bfnv, bfiter, bestCost, bestTT, bestnv,
                          bestTime, bfTime, bestiter, numfeas : INTEGER;
                    INOUT tour, bestTour, bfTour : tourType);
VAR
        i,                      {index, usually current node for moving}
        j,
        k,                      {iteration number}
        l,                      {index only}
        fhv,                    {Woodruff&Zemel 1st level hash value}
        shv,                    {Woodruff&Zemel 2nd level hash value}
        tourLen,          {entire length of time tour takes}
        ssltlc,           {steps since last tabu length change}
        escBest, {the objective value of the best of all moves}
        Dbest,            {smallest swap cost among all neighbors}
        Dbestf,           {smallest swap cost among feasible neighbors}
        chI,              {choice node initiating overall best insert move}
        chD,              {choice node receiving overall best insert move}
                    {"ch"'s may be initially set to nontabu infeasible moves
                     or infeasible moves that aspire at insert move search  }
        feasI,            {node initiating "good" feasible insert move}
        feasD,            {node receiving "good" feasible insert move}
        escI,             {node initiating "good" escape insert move}
        escD,             {node receiving "good" escape insert move}
        nodetype,         {type of node considered for insertion}
        nexttype,         {type of node next to the considered insert node}
        next2type,        {type of 2 steps from the considered insert node}
        lf,               {id of node on left}
        rt,               {id of node on right}
        d,                {index for insert DEPTH}
        dstart,           {initial value for DEPTH index in EARLY loop}
        moveVal,          {move value (curr tour to nbr), tvl + pen change}
        totNbrPen,        {total penalty for neighbor tour}
        zin,              {zin and zout update the tour hash value}
        zout,             {   for the affected nodes only}
        nvu               {# vehicles used}
```

```
        : INTEGER;

        mavg    : REAL;             {moving average of cycle length}

        tabulist : arrInt2dimType;

        list : hashListObj;{used to instantiate the array of lists}
        zArr : arrIntType;{random weights assigned to nodes}
        node : nodeType;            {used to instantialte "working" tour}

        load,
        earlymove,                  {TRUE if an early move is to be performed}
        found : BOOLEAN;            {TRUE if curr tour was visited before}

        hashcurr : hashRecord;      {curr tour's 2nd hash and other info}
        hashtbl : hashTblType;      {array of hash lists indexed by 1st hash}
        nbrtour : tourType;         {working tour for insertion operation}
        nbrPen : vrpPenType;        {penalty record of neighbor tour}
        str,
        where : STRING;             {used as diagnostic check}

CONST
{     iter penTrav  bestCost  bfCost}
format="****< *******< ********< *******<";

BEGIN
{**
ASK outstrm2 WriteString("iter tabuLen penTrav bestCost bfCost");
ASK outstrm2 WriteLn;
**}

{initialize the RTS parameters}
mavg := FLOAT(numnodes - 2);
ssltlc := 0;

{initialize tabu array to zero}
NEW(tabulist, 0..numnodes, 0..numnodes);
FOR i := 0 TO numnodes
        FOR j := 0 TO numnodes
                tabulist[i][j] := 0;
        END FOR;
END FOR;

k := 1; {first iteration}
numfeas := 0;

NEW(hashtbl, 0..HTSIZE);                {instantiate the hash table}
FOR i := 0 TO HTSIZE
        NEW(list);
        hashtbl[i] := list;
END FOR;

randWtWZ(nc, numnodes, ZRANGE, zArr);          {assign random wts to each node}

tourHVwz(numnodes, tour, zArr, shv);       {start tour's 2nd level hash val}
```

```
{Place initial tour in hash table}
{create a hashRecord}
NEW(hashcurr);

{assign the 2nd level hash value, tourCost, tvl & penalties hashcurr}
hashcurr.shv := shv;
hashcurr.cost := tourCost;
hashcurr.tvl := tvl;
hashcurr.twpen := tourPen.tw;
hashcurr.lastiter := 0;

{find the 1st level hash value for the current tour}
fhv := tourCost MOD HTSIZE;

{Add current hash record object to the linked list indexed by fhv}
ASK hashtbl[fhv] AddFirst(hashcurr);


{********** MAJOR SEARCH LOOP **************}
WHILE k <= iters

IF moveprint
ASK outstrm WriteLn;
str := "k = " + INTTOSTR(k);
ASK outstrm WriteString(str);
END IF;

  {** FIND INCUMBENT TOUR **}
  {initialize "move" parameters}
  Dbest := 999999;
  escBest := 999999;
  Dbestf := 999999;
  chI := 0;
  chD := 0;
  feasI := 0;
  feasD := 0;
  escI := 0;
  escD := 0;

  {********************************}
  {** check all LATER insertions**}
  FOR i := 1 TO numnodes-2

    {copy incumbent tour to working copy "nbr"}
    NEW(nbrtour, 0..numnodes);
    nbrtour := CLONE(tour);
    FOR l := 0 TO numnodes
          nbrtour[l] := CLONE(tour[l]);
    END FOR;

    nodetype := tour[i].type;          {determine current nodes type}

    d := 1;
    WHILE d <= DEPTH  {DEPTH loop}
```

```
IF i+d < numnodes - 1       {feasible depth?}

  {determine type of node on right}
  nexttype := tour[i+d].type;

  IF nodetype = 1 {customer node}

   {if strong TWs violated within a vehicle, move the customer
   along until a vehicle is encountered, then swap and
   "locally" update the schedule as the customer is
   swapped, and increment d as well}

   {strong TW check}
   IF (tour[i+d].ea + time[tour[i+d].id][tour[i].id])
     > tour[i].la

    WHILE nbrtour[i+d].type = 1

        lf := i+d-1;
        rt := i+d;
        SwapNode(nbrtour[lf], nbrtour[rt]);

        {local update:arr,dep,wait}
        nbrtour[lf].arr := nbrtour[lf-1].dep +
                        time[nbrtour[lf-1].id][nbrtour[lf].id];
        nbrtour[lf].dep := MAXOF(nbrtour[lf].ea, nbrtour[lf].arr);
        nbrtour[lf].wait :=nbrtour[lf].dep - nbrtour[lf].arr;

        {local update:load}
        IF nbrtour[lf-1].type = 2
         nbrtour[lf].load := nbrtour[lf].qty;
        ELSE
         nbrtour[lf].load := nbrtour[lf-1].load + nbrtour[lf].qty;
        END IF;

        d := d + 1;

        {IF with EXIT from "DEPTH" loop}
        {because if you increment to numnodes-1, don't want}
        {to do a swap with terminal depot}
        IF i+d = numnodes-1 EXIT; END IF;

    END WHILE;

   END IF;{TW check}

   {The customer is now ready to have its move evaluated:
    1 Swap it with the next node
    2 Compute the change in travel distance, and compute the
      neighbor's schedule
3 Compute the neighbor's penalty values
    4 Increase the total move value by the "costed penalties"}

   {1} SwapNode(nbrtour[i+d-1], nbrtour[i+d]);
```

```
            {2} moveValTT(i, d, numnodes, tour, nbrtour, time, moveVal);

            {3} NEW(nbrPen);
              compPens(numnodes, nbrtour, 0, nbrPen);
            {4} totNbrPen := nbrPen.tw;
              moveVal:= moveVal+TRUNC(TWPEN*FLOAT(nbrPen.tw-tourPen.tw));

                DISPOSE(nbrPen);

IF stepprint
ASK outstrm WriteLn;
str := "node " + INTTOSTR(tour[i].id) + " d = " + INTTOSTR(d) + " ";
ASK outstrm WriteString(str);
str := " moveVal = " + INTTOSTR(moveVal) + " totNbrPen = " + INTTOSTR(totNbrPen);
ASK outstrm WriteString(str);
 IF i+d < numnodes
 IF k <= tabulist[tour[i].id][i+d]
        ASK outstrm WriteString(" **TABU");
 END IF;
 END IF;
END IF;

            {END nodetype = 1 (customer)}

           ELSE {nodetype = 2, vehicle
                and vehicles are always strong TW feasible
                IF next node is a customer, move is valid}

            IF nexttype = 2 EXIT; END IF;
            {dont swap adjacent vehicles, leave "d" loop}
            {1} SwapNode(nbrtour[i+d-1], nbrtour[i+d]);
            {2} moveValTT(i, d, numnodes, tour, nbrtour, time, moveVal);

            {3} NEW(nbrPen);
              compPens(numnodes, nbrtour, 0, nbrPen);
            {4} totNbrPen := nbrPen.tw;
              moveVal:= moveVal+TRUNC(TWPEN*FLOAT(nbrPen.tw-tourPen.tw));

                DISPOSE(nbrPen);

IF stepprint
ASK outstrm WriteLn;
str := "node " + INTTOSTR(tour[i].id) + " d = " + INTTOSTR(d) + " ";
ASK outstrm WriteString(str);
str := " moveVal = " + INTTOSTR(moveVal) + " totNbrPen = " + INTTOSTR(totNbrPen);
ASK outstrm WriteString(str);
 IF i+d < numnodes
 IF k <= tabulist[tour[i].id][i+d]
        ASK outstrm WriteString(" **TABU");
 END IF;
 END IF;
END IF;

           END IF; {nodetype check}
```

```
IF totNbrPen = 0              {feasible candidate tour?}

  IF (moveVal < Dbestf)  {If this is best feasible neighbor,}
                         {AND (not tabu OR it aspires), SAVE!}
   IF ( k > tabulist[tour[i].id][i+d] )
    OR ( moveVal + penTrav < bestCost )

      Dbestf := moveVal;
      feasI := i;
      feasD := d;

   END IF; {not tabu OR aspires}
  END IF; {moveVal < DbestF}
{END totNbrPen = 0}

ELSE {candidate is infeasible}

  IF (moveVal < Dbest)  {IF this is best infeas neighbor, SAVE}

        IF ( k > tabulist[tour[i].id][i+d] )
         OR ( moveVal + penTrav < bestCost )

         Dbest := moveVal;
         chI := i;
         chD := d;

        END IF; {not tabu OR aspires}
  END IF; {moveVal < Dbest}
END IF; {infeas candidate}

{Escape Routine}
{saves the best of all neighbor moves in case all moves tabu or
 non-quality changing}
IF moveVal < escBest
  escBest := moveVal
  escI := i;
  escD := d;

END IF;{escape}

{IF only vehicle nodes are left in the tour, STOP, }
{get the next node. Compare the position to the id of the }
{node, IF equal you are at the end of the tour (Carlton, 95:5.3)}
 IF ( nbrtour[i+d+1].type = 2 )
   AND ( nbrtour[i+d+1].id = i + d + 1 )

   EXIT;  END IF;

ELSE    {i+d < numnodes - 1 (feasible DEPTH)}

   EXIT;

END IF;

d := d + 1;
```

```
    END WHILE;          {d = 1 to DEPTH}

    FOR l := 0 TO numnodes
        DISPOSE(nbrtour[l]);
    END FOR;
    DISPOSE(nbrtour);

IF stepprint
ASK outstrm WriteLn;
str := "Dbestf = " + INTTOSTR(Dbestf) + " Dbest = " + INTTOSTR(Dbest)
    + " escBest = " + INTTOSTR(escBest);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;

  END FOR;{i = 1 to numnodes-2}

  {***********************************}
  {*** check all EARLIER insertions ***}

  i := 3

  WHILE i <= numnodes-1

    earlymove := TRUE; {initially, we intend to perform a move}

    {create working copy}
    nbrtour := CLONE(tour);
    FOR l := 0 TO numnodes
        nbrtour[l] := CLONE(tour[l]);
    END FOR;

    {do not consider any d = -1 moves as they are later moves}
    d := 1;

    nodetype := tour[i].type;
    nexttype := tour[i-1].type;
    next2type := tour[i-2].type;

    IF nodetype = 2 {vehicle node}

     IF (nexttype <> 2) AND (next2type <> 2)
     {dont want adjacent vehicles or a sandwiched customer}

         SwapNode(nbrtour[i-d], nbrtour[i-d+1]);
         d := d + 1;

     ELSE

         earlymove := FALSE; {GOTO NEXT NODE}

       END IF; {nexttype or next2type = 2}

    ELSE {customer node}
```

```
{strong TW check}
IF tour[i].ea + time[tour[i].id][tour[i-1].id] <= tour[i-1].la
 {do the d = -1 swap (i and i-1)}

 SwapNode(nbrtour[i-d], nbrtour[i-d+1]);
 d := d + 1;

    ELSE {TW check NOT OK}

 {do swaps to the next earlier vehicle node}
 {stop while a customer is adjacent}
 WHILE nbrtour[i-d].type = 1
  SwapNode(nbrtour[i-d], nbrtour[i-d+1]);
  d := d + 1;

 END WHILE;

 {if we are now at start depot, GOTO NEXT NODE}
 IF i-d = 0
  earlymove := FALSE;
 END IF;


END IF; {strong TW check}

END IF; {END for customer node}

IF earlymove = TRUE

 WHILE d <= DEPTH          {DEPTH loop}

  IF i-d <= 0 {feasible DEPTH check}

   EXIT;     {avoid unnecessary loops}

  ELSE

   IF nodetype = 1
    {strong TW check}
    IF tour[i].ea + time[tour[i].id][tour[i-d].id]
      > tour[i-d].la

      {swap adjacent customers}
      WHILE nbrtour[i-d].type = 1

      SwapNode(nbrtour[i-d], nbrtour[i-d+1]);
      d := d + 1;
      END WHILE;

      {stop at node 0, GOTO NEXT NODE (i)}
      IF i-d = 0
       EXIT;
      END IF;

    END IF; {strong TW check}
```

```
        {*now evaluate neighbor tour*}

            {1} SwapNode(nbrtour[i-d], nbrtour[i-d+1]);
            {2} moveValTT(i, -d, numnodes, tour, nbrtour, time, moveVal);
            {3} NEW(nbrPen);
              compPens(numnodes, nbrtour, 0, nbrPen);
            {4} totNbrPen := nbrPen.tw;
              moveVal:= moveVal+TRUNC(TWPEN*FLOAT(nbrPen.tw-tourPen.tw));

                  DISPOSE(nbrPen);

IF stepprint
ASK outstrm WriteLn;
str := "node " + INTTOSTR(tour[i].id) + " d = " + INTTOSTR(-d) + " ";
ASK outstrm WriteString(str);
str := " moveVal = " + INTTOSTR(moveVal) + " ";
ASK outstrm WriteString(str);
 IF i+d < numnodes
 IF k <= tabulist[tour[i].id][i+d]
        ASK outstrm WriteString(" **TABU");
 END IF;
 END IF;
END IF;

       ELSE      {END for customer node, start vehicle node}

        nexttype := tour[i-d-1].type;

        {dont swap to adjacent vehicles, eval next node}
        IF (nexttype = 2)

                EXIT; {GOTO NEXT NODE (i)}

        ELSE

         {*evaluate neighbor tour*}
             {1} SwapNode(nbrtour[i-d], nbrtour[i-d+1]);

             {2} moveValTT(i, -d, numnodes, tour, nbrtour, time, moveVal);
             {3} NEW(nbrPen);
               compPens(numnodes, nbrtour, 0, nbrPen);
             {4} totNbrPen := nbrPen.tw;
               moveVal:= moveVal+TRUNC(TWPEN*FLOAT(nbrPen.tw-tourPen.tw));

                   DISPOSE(nbrPen);

IF stepprint
ASK outstrm WriteLn;
str := "node " + INTTOSTR(tour[i].id) + " d = " + INTTOSTR(-d) + " ";
ASK outstrm WriteString(str);
str := " moveVal = " + INTTOSTR(moveVal) + " ";
ASK outstrm WriteString(str);
 IF i+d < numnodes
 IF k <= tabulist[tour[i].id][i+d]
```

```
        ASK outstrm.WriteString(" **TABU");
  END IF;
  END IF;
 END IF;

             END IF; {END adjacent vehicle check}

       END IF; {END for vehicle node}

       {feasible tour?}
       IF totNbrPen = 0

        IF (moveVal < Dbestf)

         {IF not tabu OR aspires}
         IF ( k > tabulist[tour[i].id][i-d] )
          OR ( moveVal + penTrav < bestCost )

          Dbestf := moveVal;
          feasI := i;
          feasD := -d;

         END IF; {IF not tabu OR aspires}
        END IF; {moveVal < Dbestf}

       ELSE {infeasible tour}

        IF (moveVal < Dbest)

         {IF not tabu OR aspires}
         IF ( k > tabulist[tour[i].id][i-d] )
          OR ( moveVal + penTrav < bestCost )

          Dbest := moveVal
          chI := i;
          chD := -d;

         END IF; {IF not tabu OR aspires}
        END IF; {moveVal < Dbest}

       END IF; {feasible tour check}

           {Escape Routine}
           {saves the best of all neighbor moves in case all moves tabu
            or non-quality changing}
           IF moveVal < escBest
            escBest := moveVal
            escI := i;
            escD := -d;

           END IF;{escape}

      END IF; {feasible DEPTH check}

          d := d + 1;
```

```
   END WHILE; {DEPTH loop}

   END IF; {earlymove=TRUE}

   i := i + 1;

IF stepprint
ASK outstrm WriteLn;
str := "Dbestf = " + INTTOSTR(Dbestf) + " Dbest = " + INTTOSTR(Dbest)
      + " escBest = " + INTTOSTR(escBest);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;

   FOR l := 0 TO numnodes
         DISPOSE(nbrtour[l]);
   END FOR;
   DISPOSE(nbrtour);

  END WHILE; {i = 3 TO numnodes-2}

  {If feasible move found, move to it}
  IF feasI <> 0
   chI := feasI;
   chD := feasD;


  {** IF ALL MOVES ARE TABU AND NONE MEET ASPIRATION CRITERIA **}
  {                                                              }
  {  THEN SET chI AND chD TO THE BEST MOVE DISCOVERED                          }
  {     AND DECREASE THE TABU LENGTH                                   }
  {                                                              }
  {         OR IF NO MOVES ARE AVAILABLE                               }
  {*************************************************************}

  {NO MOVES ARE AVAILABLE}
  {This "degenerate" condition only occurs whenever only one
   vehicle is available and no feasible moves are available
   because of STRONG TW feasibility.  This stops all computation
   and prompts the user to restart allowing more than one vehicle.}

  ELSIF escI = 0
   ASK outstrm WriteLn;
   ASK outstrm WriteInt(k, 4);
   ASK outstrm WriteString("There are no moves available....");
   ASK outstrm WriteString("Increase the number of vehicles and try again");
   ASK outstrm WriteLn;
   EXIT;

  {ALL MOVES ARE TABU AND NONE MEET ASPIRATION CRITERIA}
  ELSIF chI = 0

ASK outstrm WriteString("All moves tabu and none meet aspiration criteria ");
ASK outstrm WriteString("at iteration: ");
ASK outstrm WriteInt(k, 4);
```

```
ASK outstrm WriteLn;

   {best of the neighbors is still moved to, tabu length adjusted}
   chI := escI;
   chD := escD;
   tabuLen := MAXOF( ROUND(FLOAT(tabuLen) * DECREASE), minTL);
  END IF;

  {** UPDATE TABU LIST AND TOUR POSITIONS **}
  {allow no "return" moves for tabuLen iterations, See Carlton '95: }
  {4.3.6. Prevents a direct (active) move back to the position }
  {which the node just moved from}

  IF chD = 1
   tabulist[tour[chI+1].id][chI+1] := k + tabuLen;
  ELSE
   tabulist[tour[chI].id][chI] := k + tabuLen;
  END IF;

  {allow no "repeat" moves for tabuLen iterations, See Carlton '95: }
  {4.3.6.  Prevents a direct (active) move back into the position }
  {into which the node is currently moving}

  tabulist[tour[chI].id][chI+chD] := k + tabuLen;

  {BEFORE the new tour is constructed, update the tour hashing value}
  {Performed exactly like a 3-opt move update, Wooruff&Zemel (93)}
  zin := 0;  zout := 0;

  i := chI;
  IF chD > 0
   j:= chI + chD;
  ELSE
   j:= chI + chD - 1;
  END IF;

  zout := (zArr[tour[i-1].id] * zArr[tour[i].id])
       + (zArr[tour[i].id] * zArr[tour[i+1].id])
       + (zArr[tour[j].id] * zArr[tour[j+1].id]);

  zin := (zArr[tour[i-1].id] * zArr[tour[i+1].id])
       + (zArr[tour[j].id] * zArr[tour[i].id])
       + (zArr[tour[i].id] * zArr[tour[j+1].id]);

  shv := shv + (zin - zout);

IF moveprint
ASK outstrm WriteLn;
str := "Move inserts node " + INTTOSTR(tour[chI].id) + " to position "
     + INTTOSTR(chI + chD);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
str := "w/ shv = " + INTTOSTR(shv);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;
```

```
  {Perform the insertion}
  insert(chI, chD, tour);

IF moveprint
qcktourFile(outstrm, tour, numnodes);
END IF;

  {*UPDATE THE NEW INCUMBENT SCHEDULE*}
  {* schedule data and tour length   *}
  IF chD > 0
        tourSched(chI, nc, numnodes, tour, time, tourLen, outstrm);
  ELSE
        tourSched(chI+chD, nc, numnodes, tour, time, tourLen, outstrm);
  END IF;

  {update penalties}
  compPens(numnodes, tour, 0, tourPen);
  tsptwPen(numnodes, tourLen, tour, tourPen, TWPEN, totPenalty,
           tourCost, penTrav, tvl);

IF moveprint
str := " and Tour Cost = " + INTTOSTR(tourCost);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
str := "Current mavg is " + REALTOSTR(mavg) + " and Steps since last TL change "
        + INTTOSTR(ssltlc) + " current tabuLen " + INTTOSTR(tabuLen);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;

  {*********}
  {CYCLE CHECK}

  fhv := tourCost MOD HTSIZE;
  lookfor(fhv, tourCost, shv, tvl, k, tourPen, hashtbl, hashcurr, found);

  {if exact match exists then we found a cycle}

  IF found = FALSE {new unfound feasible tour}
    IF totPenalty = 0
          numfeas := numfeas +1;
    END IF;

    countVeh(numnodes, tour, nvu);
    twBestTT(numnodes, totPenalty, penTrav, tvl, nvu, k, tour, bfCost, bfTT,
                bfnv, bfiter, bestCost, bestTT, bestnv, bestiter, bfTour,
                bestTour, bfTime, bestTime);
    nocycle(DECREASE, minTL, mavg, ssltlc, tabuLen, outstrm, cycleprint);

IF moveprint
str := "This tour was NOT FOUND in the hashing structure";
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;

  ELSE
    {use hashcurr to get correct "lastiter"}
    cycle(hashcurr, INCREASE, maxTL, CYMAX, k, mavg, ssltlc, tabuLen,
```

```
          outstrm, cycleprint);

IF moveprint
str := "This tour was FOUND in the hashing structure";
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;

END IF;

{**
IF moveprint
str := " "
twLoadToFile(str, outstrm, tour, nc, numnodes, tourLen, TRUE);
END IF;
**}

{*** OUTPUT("k = ", k," and bestCost = ", bestCost); ***}
{**
IF (k MOD 10) = 0
 ASK outstrm2 WriteInt(k, 4); ASK outstrm WriteString(" ");
 ASK outstrm2 WriteInt(tabuLen, 4); ASK outstrm WriteString(" ");
 ASK outstrm2 WriteInt(penTrav, 7); ASK outstrm WriteString(" ");
 ASK outstrm2 WriteInt(bestCost, 7); ASK outstrm WriteString(" ");
 ASK outstrm2 WriteInt(bestCost, 7); ASK outstrm WriteString(" ");
 ASK outstrm2 WriteLn;
END IF;
**}
  k := k + 1;

END WHILE; {* TABU SEARCH ROUTINE END *}

DISPOSE(hashtbl);
DISPOSE(tabulist);
DISPOSE(zArr);

END METHOD; {search}

END OBJECT; {reacTabuObj}


END MODULE. {Implementation}
```

Appendix C: hashMod

The "hashMod" library contains the procedures associated with using a hashing structure as the long-term memory component to a tabu search. The definition module creates the hash record and table data structures. The procedure "lookfor" compares a tour to those values in the hash table. While "randWtWZ" assigns the random weights in a form suggested by Woodruff and Zemel (1993), "tourHVwz" assigns the hash value to a given tour.

```
DEFINITION MODULE hashMod;

FROM ListMod IMPORT BasicRankedList;
FROM ListMod IMPORT BasicListObj;

FROM tabuMod IMPORT vrpPenType;
FROM tabuMod IMPORT tourType;
FROM tabuMod IMPORT arrIntType;


TYPE

hashListObj = OBJECT (BasicListObj, BasicRankedList)

END OBJECT;

hashRecord = RECORD    {The Hashing structure}

        shv,                               {2nd level tour hashing value}
        cost,                              {tour cost = tourLen + TW cost}
        tvl,                               {tour's travel time}
        twpen,                             {tour's time window penalty}
        loadpen,                    {tour's load penalty}
        lastiter   : INTEGER;       {iteration of last visit}

        next       : hashRecord;

END RECORD; {hashRecord}


hashTblType = ARRAY INTEGER OF hashListObj;

{Looks for the current tour in the hashing structure, If the tour is found,
 the pointer to the hash table is returned.  If not found, the tour is added to
 the structure and a NILREC pointer is returned}
PROCEDURE lookfor (IN fhv, tourCost, shv, tvl, iter: INTEGER;
                   IN tourPen : vrpPenType;
```

```
                    IN hashtbl : hashTblType;
                    INOUT hashcurr : hashRecord; {pointer to current record}
                    OUT found : BOOLEAN);

{Computes the Woodruff & Zemel (1993) hashing value from the sum of adjacent
 node id multiplication}
PROCEDURE tourHVwz (IN numnodes : INTEGER; IN tour : tourType;
                        IN zArr : arrIntType;
                        OUT tourHV : INTEGER);

{Assigns random weights between 1 & RANGE to the nodes for Woodruff&Zemel (1993) hashing value
calculations}
PROCEDURE randWtWZ (IN nc, numnodes, ZRANGE : INTEGER;
                        OUT zArr : arrIntType);


END MODULE.


IMPLEMENTATION MODULE hashMod;

FROM tabuMod IMPORT tourType;
FROM tabuMod IMPORT vrpPenType;
FROM tabuMod IMPORT tourType;
FROM tabuMod IMPORT arrIntType;

FROM RandMod IMPORT RandomObj;


{Looks for the current tour in the hashing structure, If the tour is found,
 the pointer to the hash table is returned.  If not found, the tour is added to
 the structure and a NILREC pointer is returned}
PROCEDURE lookfor (IN fhv, tourCost, shv, tvl, iter: INTEGER;
                    IN tourPen : vrpPenType;
                    IN hashtbl : hashTblType;
                    INOUT hashcurr : hashRecord; {pointer to current record}
                    OUT found : BOOLEAN);
VAR
        hnp : hashRecord;           {hash list pointer}
BEGIN
        found := FALSE;

        {While the list pointer is not NILREC, search the hash table to determine
         if the current tour values are equal to any stored values.  If so,
         return the pointer}

        hnp := ASK hashtbl[fhv] First;

        WHILE hnp <> NILREC
                IF (hnp.twpen = tourPen.tw)
                  AND (hnp.loadpen = tourPen.ld)
                  AND (hnp.shv = shv)
                  AND (hnp.tvl = tvl)

                        hashcurr := hnp;
```

```
                    found := TRUE;
                    EXIT;
            ELSE
                    hnp := ASK hashtbl[fhv] Next(hnp);
            END IF;
        END WHILE;

        {If not found, add to hash table}
        IF found = FALSE
            NEW(hashcurr);
            hashcurr.shv := shv;
            hashcurr.cost := tourCost;
            hashcurr.tvl := tvl;
            hashcurr.twpen := tourPen.tw;
            hashcurr.loadpen := tourPen.ld;
            hashcurr.lastiter := iter;

            ASK hashtbl[fhv] AddFirst(hashcurr);
            {use AddFirst to prevent another run through the list}
        END IF;

END PROCEDURE; {lookfor}

{Computes the Woodruff & Zemel (1993) hashing value from the sum of adjacent
 node id multiplication}
PROCEDURE tourHVwz (IN numnodes : INTEGER; IN tour : tourType;
                    IN zArr : arrIntType;
                    OUT h3 : INTEGER);
VAR
        i : INTEGER;
BEGIN
        h3 := 0;

        FOR i := 0 TO numnodes-1
            h3 := h3 + zArr[tour[i].id] * zArr[tour[i+1].id];
        END FOR;

END PROCEDURE; {tourHVwz}

{Assigns random weights between 1 & RANGE to the nodes for Woodruff&Zemel (1993) hashing value
calculations}
PROCEDURE randWtWZ (IN nc, numnodes, ZRANGE : INTEGER;
                    OUT zArr : arrIntType);
VAR
        i : INTEGER;
        randObj : RandomObj;
BEGIN
        NEW(zArr, 0..numnodes);
        NEW(randObj);

        FOR i := 0 TO nc
            zArr[i] := ABS( ASK randObj UniformInt(1, ZRANGE) );

{* OUTPUT("zArr[",i,"] = ", zArr[i]); *}
```

```
        END FOR;
        DISPOSE(randObj);

        {vehicle nodes same as depot}
        FOR i := nc+1 TO numnodes
                zArr[i] := zArr[0];

{* OUTPUT("zArr[",i,"] = ", zArr[i]); *}

        END FOR;

END PROCEDURE; {tourHVwz}


END MODULE.
```

Appendix D: bestSolnMod

The "bestSolnMod" library contains only one procedure intended for Carlton's (1995) reactive tabu search. This procedure, twBestTT, tracks the best travel times and tour costs and saves the corresponding tours. A separate library was created because GVRP research is known to use many different forms of the objective function. The implementation module follows.

```
IMPLEMENTATION MODULE bestSolnMod;

FROM OSMod IMPORT SystemTime;
        {VRP data types}
FROM tabuMod IMPORT tourType;

FROM tabuMod IMPORT countVeh;

{best Travel Time, lowest number of vehicles}
{Retains the feasible solution having the shortest travel time and with the
 shortest travel time has the shortest completion time
        first saves tour with shortest travel time
        ties broken by shortest completion time
        and/or number of vehicles used}
PROCEDURE twBestTT (IN numnodes,
                        totPenalty,         {total penalty}
                        penTrav,            {current tour: penalty + tvl}
                        tvl,                {current tour: travel time}
                        nvu,                {number of vehicles used}
                        iter                {current iteration number}
                            : INTEGER;
                    IN tour : tourType;     {current tour}
                    INOUT bfCost, bfTT, {best feas cost & tvl time}
                          bfnv,         {best feas num vehs used}
                          bfiter,       {iter # when best feas found}
                          bestCost,         {best overall penalty + TT}
                          bestTT,           {best overall travel time}
                          bestnv,           {best number of vehs used}
                          bestiter          {iter # when best ovrall found}
                            : INTEGER;
                    INOUT bfTour, bestTour : tourType;
                    INOUT bfTime, bestTime : INTEGER );
VAR
        i,
        currtime  : INTEGER;    {current clock time of search}
BEGIN
        currtime := SystemTime();

        {save the tour if it is the best ever found}
```

```
IF penTrav < bestCost

  bestTT := tvl;
  bestCost := penTrav;
  bestTime := currtime;
  bestiter := iter;
  FOR i := 0 TO numnodes
          bestTour[i] := CLONE(tour[i]);
  END FOR;
  bestnv := nvu;

ELSIF (penTrav = bestCost) AND (nvu < bestnv)

  bestTime := currtime;
  bestTT := tvl;
  bestiter := iter;
  FOR i := 0 TO numnodes
          bestTour[i] := CLONE(tour[i]);
  END FOR;
  bestnv := nvu;

END IF;

{feasible checks}
IF (tvl > bfTT) OR (totPenalty > 0)
          RETURN;
ELSIF (tvl < bfTT) AND (totPenalty = 0)
          bfTime := currtime;
          bfCost := penTrav;
          bfTT := tvl;
          bfiter := iter;
          FOR i := 0 TO numnodes
            bfTour[i] := CLONE(tour[i]);
          END FOR;
          bfnv := nvu;
          RETURN;
ELSIF (tvl = bfTT) AND ((penTrav < bfCost) OR (nvu < bfnv))
          bfTime := currtime;
          bfCost := penTrav;
          bfTT := tvl;
          bfiter := iter;
          FOR i := 0 TO numnodes
            bfTour[i] := CLONE(tour[i]);
          END FOR;
          bfnv := nvu;
          RETURN;
END IF;

RETURN;

END PROCEDURE; {twbestTT}

END MODULE.
```

Appendix E: Mtsptw

The main module MtsptwMod is used for mTSPTW problems. It may also be used for TSP and mTSP problems. As written, it can work on much of the Solomon datasets (1987) in one run.

```
MAIN MODULE tsptw;

FROM IOMod IMPORT StreamObj, ALL FileUseType, ReadKey;
FROM OSMod IMPORT SystemTime;

FROM tsptwMod IMPORT timeMatrixObj;
FROM twReduceMod IMPORT twReductionObj;
FROM tsptwMod IMPORT startTourObj;
FROM tsptwMod IMPORT reacTabuObj;

FROM tabuMod IMPORT coordArrType;
FROM tabuMod IMPORT tourType;
FROM tabuMod IMPORT vrpPenType;
FROM tabuMod IMPORT arrInt2dimType;
FROM tabuMod IMPORT arrIntType;
FROM tabuMod IMPORT arrRealType;

FROM tabuMod IMPORT twLoadToFile;
FROM tabuMod IMPORT timeToFile;

VAR
        timeMatrix : timeMatrixObj;
        twReduce : twReductionObj;
        startTour : startTourObj;
        rts : reacTabuObj;

        instrm,
        outstrm,
        outstrm2 : StreamObj;

        factor,    {used to convert the coordinates to integer values}
        TWPEN,          {Penalty weight assigned to the sum of late arr TW violations}
        INCREASE,       {RTS parameter: mult. factor to decrease tabu length}
        DECREASE        {RTS parameter: mult. factor to increase tabu length}
                : REAL;

        i, j, k,
        endnum,
        maxtime,        {max possible time of arrival to any node, for time read}
        numcycles,      {number of TW reduction cycles wanted}
        numchanges,     {number of TWs reduced by TW reduction Obj}
        numnodes,       {number of nodes in the directed graph}
        nv,             {number of vehicles}
        nc,             {number of targets/customers}
        gamma,          {arbitrary cost assigned to the use of each vehicle}
```

```
iters,          {number of Tabu Search Iterations per problem}
tourLen, {Length of tour in time}
tvl,            {travel time of tour}
totPenalty,     {Total Penalty assigned to current tour}
tourCost,       {tour Length + Time Window Cost}
penTrav,        {tourCost - totWait == travel time + TW penalty}
bfTourCost,     {lowest tourCost found for a feasible tour}
bestCost,       {lowest tourCost found for a any tour}
bestTT,         {lowest travel time found for a any tour}
bestnv,         {# vehs used by best overall tour}
bfTT,           {lowest travel time found for feasible tour}
bfnv,           {# vehs used by best feas tour}
bfiter,    {iteration # when best feasible tour found}
tourhv,         {tour's hashing value}
bestiter, {iteration the best Tour found}
bestTime,       {Time the best Tour found}
bestTimeF,      {Time the best feasible Tour found}
numfeas,        {number of feasible solns found}
startTime,      {start Time (after time matrix, before TW reductions)}
stopTime,       {stop Time (after last iteration)}
duration,{stopTime - startTime}
DEPTH,          {depth of nodes we look for insert moves}
ZRANGE,                 {upper bound on random integer weights assigned to nodes}
HTSIZE,                 {size of hash table array}
CYMAX,                  {max cyleLength used to alter mavg}
tabuLen, {current length of tabu tenure}
minTL,          {minimum Tabu Length}
maxTL           {maximum Tabu Length}
        : INTEGER;

outfile,  {name of output file}
where,          {where in the code?}
file, filein,
filebegin,
fileout2,
fileout : STRING;       {filenames}

loadprint,              {print load on vehicles}
stepprint,              {print each move evaluation}
moveprint,              {print every insert move made by RTS}
startprint,             {print starting tour and tw reduction steps}
cycleprint,             {print hash results}
twrdprint : BOOLEAN;    {print tw reduction steps}

coord     : coordArrType;           {coordinates array}
bestTourF,                          {best feasible tour found}
bestTour,                           {node array holding best tour}
tour      : tourType;               {node array holding the tour}
tourPen   : vrpPenType;         {record of curr tour penalties}
s         : arrIntType;         {array of service times}
time      : arrInt2dimType;         {time/dist matrix}
m         : arrIntType;         {array of TW midpoints}

BEGIN
```

```
        OUTPUT(" ");
        OUTPUT("Please input the number of vehicles");
        INPUT(nv);

        OUTPUT(" ");
        OUTPUT("Please input the number of tabu search iterations");
        OUTPUT("you would like to step through.");
        INPUT(iters);

        OUTPUT(" ");
        OUTPUT("Please input the problem to work on:");
        INPUT(file);

        OUTPUT(" ");
        OUTPUT("Please input the factor necessary to convert");
        OUTPUT("the data coordinates to integer quantities,");
        OUTPUT("such as 1, 10, 100, etc.");
        INPUT(factor);

{**
        OUTPUT(" ");
        OUTPUT("Please input the number of time window reduction cycles");
        OUTPUT("you would like to step through (2 or 3 usually fine).");
        INPUT(numcycles);
**}

{** LOOP THRU MULTIPLE FILES ***

FOR j := 1 TO 6
 IF j = 1
        filebegin := "C10";
        endnum := 9;
 ELSIF j = 2
        filebegin := "R10";
        endnum := 12;
 ELSIF j = 3
        filebegin := "C20";
        endnum := 8;

 ELSIF j = 4
        filebegin := "R20";
        endnum := 11;
 ELSIF j = 5
        filebegin := "RC10";
        endnum := 8;
 ELSIF j = 6
        filebegin := "RC20";
        endnum := 8;
 END IF;

 FOR k := 1 TO endnum

        file := filebegin + INTTOSTR(k);
***}
        filein := file + ".DAT";
```

```
        fileout := file + ".OUT";

{**     fileout2 := file + "Rslt.OUT";
 **}

OUTPUT(file);
OUTPUT(filein);
OUTPUT(fileout);
{*OUTPUT(fileout2);
*}

        {INITIALIZE}
        startprint := FALSE;        {print starting tour}
        timeprint := FALSE;         {print time matrix}
        stepprint := FALSE;         {print each RTS step eval}
        moveprint := FALSE;         {print each RTS insert move}
        twrdprint := FALSE;         {print TW reduction steps}
        cycleprint := FALSE;        {print cycle/nocycle steps}
        loadprint := FALSE;         {print quantity & vehicle loads}

{*} {denotes a parameter setting}
{*      nv := 10;           *}
{*      factor := 10.0;     *}
{*      numcycles := 3;     *}
{*      iters := 1000;  *}
{*}     TWPEN := 1.0;
{*}     gamma := 0;

{*}     INCREASE := 1.2;
{*}     DECREASE := 0.9;
{*}     CYMAX := 50;
{*}     HTSIZE := 1009;             {**was 1009 by Carlton**}
{*}     ZRANGE := 5003;                     {**was 131073 by Carlton**}
{*}     minTL := 5;
{*}     maxTL := 2000;

        NEW(instrm);                                    {open problem file}
        ASK instrm Open(filein, Input);

        NEW(outstrm);
        ASK outstrm Open(fileout, Output);
{**
        NEW(outstrm2);
        ASK outstrm2 Open(fileout2, Output);
**}
        NEW(timeMatrix);                                {calc time/dist matrix}


        {reads Carlton file, finds nc, inits coord & tour}
        ASK timeMatrix readTSPTW(instrm, nc, numnodes, factor, nv,
                                coord, tour, s);

        {compute time matrix}
        ASK timeMatrix timeMatrix(nc, numnodes, gamma, factor,
                                tour, coord, time, s);
```

```
{***
        where := "timeMatrix complete";
        timeToFile(where, outstrm, time, numnodes);
***}

{*}     DEPTH := nc+nv-1;
{*}     tabuLen := MINOF(30, nc+nv-1);

        ASK instrm Close;          DISPOSE(instrm);
        DISPOSE(timeMatrix);
{***
        NEW(twReduce);             {reduce time windows}
                ASK twReduce rdWindow(outstrm, nc, numnodes, numcycles,
                                numchanges, time, tour, twrdprint);
        DISPOSE(twReduce);

        where := "TW reduction complete";
        twLoadToFile(where, outstrm, tour, nc, numnodes, tourLen, factor, loadprint);
***}
        NEW(m, 1..nc);
        FOR j := 1 TO nc
                m[j] := 0;
        END FOR;

        NEW(startTour); {find intial tour}
        ASK startTour startTour(nv, nc, time, tour, tourLen,
                                totPenalty, tourhv, startTime, m, outstrm);

        IF startprint
        where := "startTour complete";
        twLoadToFile(where, outstrm, tour, nc, numnodes, tourLen, factor, loadprint);
        END IF;

        ASK startTour startPenBest(numnodes, tvl, tourLen, tour, TWPEN,
                                totPenalty, penTrav, tourCost, tourPen,
                                bfiter, bfTourCost, bfTT, bfnv, bestiter,
                                bestCost, bestTT, bestnv, bestTimeF, bestTime,
                                bestTour, bestTourF);

ASK outstrm WriteString("totPen penTrav tourCost bfiter biter");
ASK outstrm WriteString(" bCost bestTT bfCost bfTT");
ASK outstrm WriteLn;
ASK outstrm WriteInt(totPenalty,6);ASK outstrm WriteInt(penTrav,8);
ASK outstrm WriteInt(tourCost,9);ASK outstrm WriteInt(bfiter,7);
ASK outstrm WriteInt(bestiter,6);ASK outstrm WriteInt(bestCost,6);
ASK outstrm WriteInt(bestTT,7);
ASK outstrm WriteInt(bfTourCost,7);ASK outstrm WriteInt(bfTT,5);
ASK outstrm WriteLn; ASK outstrm WriteLn;

        IF startprint
        where := "startTour & startPen complete";
        ASK outstrm WriteString(where); ASK outstrm WriteLn;
        ASK outstrm WriteString("tourLen: "); ASK outstrm WriteInt(tourLen, 3);
        ASK outstrm WriteString(" Total Penalty: ");
        ASK outstrm WriteInt(totPenalty, 4);
```

```
ASK outstrm WriteLn;
ASK outstrm WriteString(" TourCost: ");
ASK outstrm WriteInt(tourCost, 4);
ASK outstrm WriteString(" Best Cost: ");
ASK outstrm WriteInt(bestCost, 4);
ASK outstrm WriteString(" Best Travel Time: ");
ASK outstrm WriteInt(bestTT, 4);
ASK outstrm WriteLn;
END IF;

DISPOSE(startTour);

NEW(rts);
{conduct RTS}
ASK rts search(TWPEN, INCREASE, DECREASE, HTSIZE, CYMAX, ZRANGE, DEPTH,
               minTL, maxTL, tabuLen, iters, nc, numnodes,
               outstrm, outstrm2, tourPen, time, stepprint,
               moveprint, cycleprint, tourCost, penTrav,
               totPenalty, tvl, bfTourCost, bfTT, bfnv, bfiter,
               bestCost, bestTT, bestnv, bestTime, bestTimeF,
               bestiter, numfeas, tour, bestTour, bestTourF);
DISPOSE(rts);

stopTime := SystemTime();
duration := stopTime - startTime;

where := "Search complete: BEST TOUR";
twLoadToFile(where,outstrm, bestTour, nc, numnodes, bestTT, factor,loadprint);

ASK outstrm WriteString("# vehicles used = ");
ASK outstrm WriteInt(bestnv, 2); ASK outstrm WriteLn;
ASK outstrm WriteString("Best solution found after ");
ASK outstrm WriteString(INTTOSTR(bestTime-startTime)+" secs");
ASK outstrm WriteLn;
ASK outstrm WriteString("on Iteration: "+ INTTOSTR(bestiter));
ASK outstrm WriteLn;

IF bfiter > 0
  where := "Search complete: BEST FEASIBLE TOUR";
  twLoadToFile(where, outstrm, bestTourF, nc, numnodes, bfTT, factor,
           loadprint);

  ASK outstrm WriteString("# vehicles used = ");
  ASK outstrm WriteInt(bfnv, 2); ASK outstrm WriteLn;

  ASK outstrm WriteString("Best Feasible solution found after ");
  ASK outstrm WriteString(INTTOSTR(bestTimeF-startTime)+" secs");
  ASK outstrm WriteLn;
  ASK outstrm WriteString("on Iteration: "+ INTTOSTR(bfiter));
  ASK outstrm WriteLn;
END IF;

ASK outstrm WriteLn;
ASK outstrm WriteString("Time of Search: "+INTTOSTR(duration) );
ASK outstrm WriteString(" secs"); ASK outstrm WriteLn;
```

```
        ASK outstrm Close;
{**     ASK outstrm2 Close;  **}

        DISPOSE(outstrm);
{**     DISPOSE(outstrm2);  **}
        DISPOSE(m); DISPOSE(s);
        DISPOSE(time);
        DISPOSE(coord);
        DISPOSE(tourPen);
        DISPOSE(tour);
        DISPOSE(bestTour);
        DISPOSE(bestTourF);

{**
 END FOR; {k, 1 to endnum}
END FOR; {j, file group}
**}

END MODULE; {MAIN}

{*****
        {read in time matrix directly -- for TSP problems}
        maxtime := 9999;

        ASK timeMatrix readTSP(instrm, nc, numnodes, nv, maxtime,
                               tour, bestTour, bestTourF, time);

        ASK timeMatrix readTime(instrm, gamma, nv, maxtime, nc, numnodes,
                               factor, tour, bestTourF, bestTour, time);

        {reads Carlton file, finds nc, inits coord & tour}
        ASK timeMatrix readTSPTW(instrm, nc, numnodes, factor, nv,
                                coord, tour, s);

        {compute time matrix}
        ASK timeMatrix timeMatrix(nc, numnodes, gamma, tour, coord, time, s);
*****}
```

Appendix F: uavMod

The library "uavMod" contains the objects, methods, and procedures needed for the UAV problem. Most "tsptwMod" code does not need to be rewritten for random winds or service times, but threats add expected coverage to the objective function making a rewrite necesssary for any code related to the objective function.

```
IMPLEMENTATION MODULE uavMod;

FROM IOMod IMPORT StreamObj, ALL FileUseType;
FROM MathMod IMPORT SQRT, ACOS, COS, SIN, pi;
FROM OSMod IMPORT SystemTime;
        {VRP data types}
FROM tabuMod IMPORT arrInt2dimType;
FROM tabuMod IMPORT arrReal2dimType;
FROM tabuMod IMPORT arrIntType;
FROM tabuMod IMPORT arrRealType;
FROM tabuMod IMPORT coordType;
FROM tabuMod IMPORT coordArrType;
FROM tabuMod IMPORT nodeType;
FROM tabuMod IMPORT tourType;
FROM tabuMod IMPORT vrpPenType;
        {output stuff}
FROM tabuMod IMPORT qcktourFile;
FROM tabuMod IMPORT twLoadToFile;
        {Move/order/search stuff}
FROM tabuMod IMPORT SwapInt;
FROM tabuMod IMPORT SwapNode;
FROM tabuMod IMPORT insert;
FROM tabuMod IMPORT moveValTT; {used to update travel info}
FROM tabuMod IMPORT countVeh;
        {schedule, penalty, and hash stuff}
FROM tabuMod IMPORT tourSched;
FROM tabuMod IMPORT compPens;
FROM tabuMod IMPORT tsptwPen;  {TSPTW version}
        {hashing stuff}
FROM hashMod IMPORT hashListObj;
FROM hashMod IMPORT hashRecord;
FROM hashMod IMPORT hashTblType;
FROM hashMod IMPORT randWtWZ;
FROM hashMod IMPORT tourHVwz;
FROM hashMod IMPORT lookfor;
        {reaction stuff}
FROM tabuMod IMPORT nocycle;
FROM tabuMod IMPORT cycle;


OBJECT timeMatrixObj;
```

{Reads in the the prevalent wind vector, x,y coordinates and time window file and calculates the time between each node. Does not assume the problem is symmetric, but makes it so}

```
{Reads in a UAV problem: the number of targets, the probablities of
 survival, and the target coordinates}
ASK METHOD readUAV(IN instrm : StreamObj;
                   OUT nc, numnodes : INTEGER;
                   IN factor : REAL;           {coord conversion factor}
                   IN nv : INTEGER;
                   OUT psurv : arrRealType; {prob of survival at each target}
                   OUT coord : coordArrType; {coordinate array}
                   OUT tour : tourType;        {intial tour}
                   OUT s : arrIntType;         {time of service at each target}
                   IN outstrm : StreamObj;
                   IN print : BOOLEAN);
VAR
    i, id,
    qty : INTEGER;            {quantity demanded at each node}
    xcoord, ycoord,           {coordinates from data file}
    servtime,                 {service time at node}
    late,                     {late start to TW}
    early    : REAL;          {early start to TW}
    position : coordType;     {record to instantiate array of coord}
    node     : nodeType;      {record to instantiate array of nodes}
    str : STRING;
BEGIN
    ASK instrm ReadInt(nc);          {read in # customers}

    numnodes := nc + nv;      {# nodes in directed graph}

    NEW(tour, 0..numnodes); {initialize array of nodes}
                            {node 0=depot, nc cust node}
                            {nodes > nc are vehicle}
                            {nv-1 vehicle nodes, 0 is 1st vehicle}
                            {numnodes = terminal depot}
    FOR i := 0 TO numnodes
        NEW(node);                   {instantiate each node}
        tour[i] := node;    {place each node in array}

        tour[i].id := i;             {set node id}

        IF (i = 0) OR (i > nc)              {set node types}
            tour[i].type := 2;  {2=veh node}
        ELSE
            tour[i].type := 1;  {1=cust node}
        END IF;
    END FOR;

    NEW(coord, 0..nc);        {initialize array of positions}
    NEW(s, 0..nc);            {initialize service time array}
    NEW(psurv, 0..nc);        {initialize prob of survival array}

    FOR i := 0 TO nc
        NEW(position);               {instantiate each record}
```

```
                coord[i] := position;          {place record in array}
            END FOR;

            {read in depot node}
            ASK instrm ReadReal(xcoord);
            ASK instrm ReadReal(ycoord);
            ASK instrm ReadInt(qty);
            ASK instrm ReadReal(early);
            ASK instrm ReadReal(late);
            ASK instrm ReadReal(servtime);
            ASK instrm ReadReal(psurv[0]);

            coord[0].x := xcoord;
            coord[0].y := ycoord;
            s[0] := TRUNC(factor * servtime);
            tour[0].ea := TRUNC(factor*early);    {use Int times}
            tour[0].la := TRUNC(factor*late);
IF print
ASK outstrm WriteString("READ UAV INFO"); ASK outstrm WriteLn;
str := "0: x = " + INTTOSTR(coord[0].x) + " ea = " + INTTOSTR(tour[0].ea);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;
            {read in customer nodes}
            FOR i := 1 TO nc
                ASK instrm ReadReal(xcoord);
                ASK instrm ReadReal(ycoord);
                ASK instrm ReadInt(tour[i].qty);
                ASK instrm ReadReal(early);
                ASK instrm ReadReal(late);
                ASK instrm ReadReal(servtime);
                ASK instrm ReadReal(psurv[i]);

                coord[i].x := xcoord;
                coord[i].y := ycoord;
                s[i] := TRUNC(factor * servtime);
                tour[i].ea := TRUNC(factor*early);    {use Int times}
                tour[i].la := TRUNC(factor*late);
IF print
str := INTTOSTR(i)+": x = " + INTTOSTR(coord[i].x) + " ea = " + INTTOSTR(tour[i].ea);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;
            END FOR;

            {initialize depot node}
            tour[0].type := 2;
            tour[0].arr := tour[0].ea;
            tour[0].dep := tour[0].ea;
            tour[0].wait := 0;
            tour[0].load := 0;

            {initialize vehicle nodes and terminal depot}
            FOR i := nc+1 TO numnodes
                tour[i] := CLONE(tour[0]);
                tour[i].id := i;
            END FOR;
```

```
END METHOD; {readUAV}

{Reads in a coords in miles scenario with Service time ranges and psurv}
ASK METHOD readUAVloiter(IN instrm : StreamObj;
                        OUT nc, numnodes : INTEGER;
                        IN factor : REAL;          {coord conversion factor}
                        IN nv : INTEGER;
                        OUT psurv : arrRealType; {prob of survival at each target}
                        OUT coord : coordArrType; {coordinate array}
                        OUT tour : tourType;       {intial tour}
                        OUT slo, shi : arrIntType; {service range at each target}
                        IN outstrm : StreamObj;
                        IN print : BOOLEAN);
VAR
    i, id,
    qty : INTEGER;              {quantity demanded at each node}
    xcoord, ycoord,             {coordinates from data file}
    servlo, servhi,             {service time range at node}
    late,                       {late start to TW}
    early    : REAL;            {early start to TW}
    position : coordType;       {record to instantiate array of coord}
    node     : nodeType;        {record to instantiate array of nodes}
    str : STRING;
BEGIN
    ASK instrm ReadInt(nc);         {read in # customers}

    numnodes := nc + nv;        {# nodes in directed graph}

    NEW(tour, 0..numnodes); {initialize array of nodes}
                            {node 0=depot, nc cust node}
                            {nodes > nc are vehicle}
                            {nv-1 vehicle nodes, 0 is 1st vehicle}
                            {numnodes = terminal depot}
    FOR i := 0 TO numnodes
        NEW(node);                  {instantiate each node}
        tour[i] := node;    {place each node in array}

        tour[i].id := i;            {set node id}

        IF (i = 0) OR (i > nc)              {set node types}
            tour[i].type := 2;  {2=veh node}
        ELSE
            tour[i].type := 1;  {1=cust node}
        END IF;
    END FOR;

    NEW(coord, 0..nc);          {initialize array of positions}
    NEW(shi, 0..nc);
    NEW(slo, 0..nc);            {initialize service time array}
    NEW(psurv, 0..nc);          {initialize prob of survival array}

    FOR i := 0 TO nc
        NEW(position);              {instantiate each record}
        coord[i] := position;       {place record in array}
```

```
                END FOR;

                {read in depot node}
                ASK instrm ReadReal(xcoord);
                ASK instrm ReadReal(ycoord);
                ASK instrm ReadInt(qty);
                ASK instrm ReadReal(early);
                ASK instrm ReadReal(late);
                ASK instrm ReadReal(servlo);
                ASK instrm ReadReal(servhi);
                ASK instrm ReadReal(psurv[0]);

                coord[0].x := xcoord;
                coord[0].y := ycoord;
                slo[0] := TRUNC(factor * servlo);
                shi[0] := TRUNC(factor * servhi);
                tour[0].ea := TRUNC(factor*early);    {use Int times}
                tour[0].la := TRUNC(factor*late);
IF print
ASK outstrm WriteString("READ UAV INFO"); ASK outstrm WriteLn;
str := "0: x = " + INTTOSTR(coord[0].x) + " ea = " + INTTOSTR(tour[0].ea);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;
                {read in customer nodes}
                FOR i := 1 TO nc
                    ASK instrm ReadReal(xcoord);
                    ASK instrm ReadReal(ycoord);
                    ASK instrm ReadInt(tour[i].qty);
                    ASK instrm ReadReal(early);
                    ASK instrm ReadReal(late);
                    ASK instrm ReadReal(servlo);
                    ASK instrm ReadReal(servhi);
                    ASK instrm ReadReal(psurv[i]);

                    coord[i].x := xcoord;
                    coord[i].y := ycoord;
                    slo[i] := TRUNC(factor * servlo);
                    shi[i] := TRUNC(factor * servhi);
                    tour[i].ea := TRUNC(factor*early);    {use Int times}
                    tour[i].la := TRUNC(factor*late);
IF print
str := INTTOSTR(i)+": x = " + INTTOSTR(coord[i].x) + " ea = " + INTTOSTR(tour[i].ea);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;
                END FOR;

                {initialize depot node}
                tour[0].type := 2;
                tour[0].arr := tour[0].ea;
                tour[0].dep := tour[0].ea;
                tour[0].wait := 0;
                tour[0].load := 0;

                {initialize vehicle nodes and terminal depot}
                FOR i := nc+1 TO numnodes
```

```
            tour[i] := CLONE(tour[0]);
            tour[i].id := i;
        END FOR;

END METHOD; {readUAVloiter}

{Reads in a Latitude and Longitude scenario: the number of targets,
 the probablities of survival, and the target coordinates}
ASK METHOD readLatLong(IN instrm : StreamObj;
                        OUT nc, numnodes : INTEGER;
                        IN factor : REAL;          {coord conversion factor}
                        IN nv : INTEGER;
                        OUT coord : coordArrType; {coordinate array}
                        OUT tour : tourType;       {intial tour}
                        OUT s : arrIntType;        {time of service at each target}
                        IN outstrm : StreamObj;
                        IN print : BOOLEAN);
VAR
        i, id,
        qty : INTEGER;             {quantity demanded at each node}
        latDeg, longDeg,
        latMin, longMin, {Latitude and Longitude units}
        latSec, longSec,
        servtime,                  {service time at node}
        late,                      {late start to TW}
        early     : REAL;          {early start to TW}
        position : coordType;      {record to instantiate array of coord}
        node      : nodeType;      {record to instantiate array of nodes}
        str : STRING;
BEGIN
        ASK instrm ReadInt(nc);          {read in # customers}

        numnodes := nc + nv;       {# nodes in directed graph}

        NEW(tour, 0..numnodes); {initialize array of nodes}
                                   {node 0=depot, nc cust node}
                                   {nodes > nc are vehicle}
                                   {nv-1 vehicle nodes, 0 is 1st vehicle}
                                   {numnodes = terminal depot}
        FOR i := 0 TO numnodes
            NEW(node);                   {instantiate each node}
            tour[i] := node;   {place each node in array}

            tour[i].id := i;             {set node id}

            IF (i = 0) OR (i > nc)               {set node types}
                tour[i].type := 2;  {2=veh node}
            ELSE
                tour[i].type := 1;  {1=cust node}
            END IF;
        END FOR;

        NEW(coord, 0..nc);         {initialize array of positions}
        NEW(s, 0..nc);             {initialize service time array}
```

```
            FOR i := 0 TO nc
                NEW(position);              {instantiate each record}
                coord[i] := position;       {place record in array}
            END FOR;

IF print
ASK outstrm WriteString("READ UAV INFO"); ASK outstrm WriteLn;
str := "0: x = " + INTTOSTR(coord[0].x) + " ea = " + INTTOSTR(tour[0].ea);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;
            {read in customer nodes}
            FOR i := 0 TO nc
             ASK instrm ReadReal(latDeg);
             ASK instrm ReadReal(latMin);
             ASK instrm ReadReal(latSec);
             ASK instrm ReadReal(longDeg);
             ASK instrm ReadReal(longMin);
             ASK instrm ReadReal(longSec);
             ASK instrm ReadReal(early);
             ASK instrm ReadReal(late);
             ASK instrm ReadReal(servtime);

             coord[i].x := (latDeg + latMin /60.0 + latSec / 3600.0) / 57.2958;
             coord[i].y := (longDeg+longMin /60.0 + longSec/ 3600.0) / 57.2958;

             s[i] := TRUNC(factor * servtime);
             tour[i].ea := TRUNC(factor*early);   {use Int times}
             tour[i].la := TRUNC(factor*late);
IF print
str := INTTOSTR(i)+": x = " + INTTOSTR(coord[i].x) + " ea = " + INTTOSTR(tour[i].ea);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;
            END FOR;

            {initialize depot node}
            tour[0].type := 2;
            tour[0].arr := tour[0].ea;
            tour[0].dep := tour[0].ea;
            tour[0].wait := 0;
            tour[0].load := 0;

            {initialize vehicle nodes and terminal depot}
            FOR i := nc+1 TO numnodes
                tour[i] := CLONE(tour[0]);
                tour[i].id := i;
            END FOR;

        END METHOD; {readLatLong}

        {Reads in a Latitude and Longitude scenario with variable Service Times}
        ASK METHOD readLatLongLoiter(IN instrm : StreamObj;
                            OUT nc, numnodes : INTEGER;
                            IN factor : REAL;           {coord conversion factor}
                            IN nv : INTEGER;
                            OUT coord : coordArrType; {coordinate array}
```

```
                              OUT tour : tourType;          {intial tour}
                              OUT slo, shi : arrIntType; {target service ranges}
                              IN outstrm : StreamObj;
                              IN print : BOOLEAN);
        VAR
            i, id,
            qty : INTEGER;              {quantity demanded at each node}
            latDeg, longDeg,
            latMin, longMin, {Latitude and Longitude units}
            latSec, longSec,
            servlo, servhi,             {service ranges at node}
            late,                       {late start to TW}
            early     : REAL;           {early start to TW}
            position : coordType;       {record to instantiate array of coord}
            node      : nodeType;       {record to instantiate array of nodes}
            str : STRING;
        BEGIN
            ASK instrm ReadInt(nc);             {read in # customers}

            numnodes := nc + nv;        {# nodes in directed graph}

            NEW(tour, 0..numnodes); {initialize array of nodes}
                                        {node 0=depot, nc cust node}
                                        {nodes > nc are vehicle}
                                        {nv-1 vehicle nodes, 0 is 1st vehicle}
                                        {numnodes = terminal depot}
            FOR i := 0 TO numnodes
                NEW(node);                      {instantiate each node}
                tour[i] := node;    {place each node in array}

                tour[i].id := i;                {set node id}

                IF (i = 0) OR (i > nc)                  {set node types}
                    tour[i].type := 2;  {2=veh node}
                ELSE
                    tour[i].type := 1;  {1=cust node}
                END IF;
            END FOR;

            NEW(coord, 0..nc);          {initialize array of positions}
            NEW(slo, 0..nc);            {initialize service time arrays}
            NEW(shi, 0..nc);

            FOR i := 0 TO nc
                NEW(position);                  {instantiate each record}
                coord[i] := position;           {place record in array}
            END FOR;

    IF print
    ASK outstrm WriteString("READ UAV INFO"); ASK outstrm WriteLn;
    str := "0: x = " + INTTOSTR(coord[0].x) + " ea = " + INTTOSTR(tour[0].ea);
    ASK outstrm WriteString(str); ASK outstrm WriteLn;
    END IF;
            {read in customer nodes}
            FOR i := 0 TO nc
```

```
                ASK instrm ReadReal(latDeg);
                ASK instrm ReadReal(latMin);
                ASK instrm ReadReal(latSec);
                ASK instrm ReadReal(longDeg);
                ASK instrm ReadReal(longMin);
                ASK instrm ReadReal(longSec);
                ASK instrm ReadReal(early);
                ASK instrm ReadReal(late);
                ASK instrm ReadReal(servlo);
                ASK instrm ReadReal(servhi);

                coord[i].x := (latDeg + latMin /60.0 + latSec / 3600.0) / 57.2958;
                coord[i].y := (longDeg+longMin /60.0 + longSec/ 3600.0) / 57.2958;

                slo[i] := TRUNC(factor * servlo);
                shi[i] := TRUNC(factor * servhi);
                tour[i].ea := TRUNC(factor*early);    {use Int times}
                tour[i].la := TRUNC(factor*late);
IF print
str := INTTOSTR(i)+": x = " + INTTOSTR(coord[i].x) + " ea = " + INTTOSTR(tour[i].ea);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;
            END FOR;

            {initialize depot node}
            tour[0].type := 2;
            tour[0].arr := tour[0].ea;
            tour[0].dep := tour[0].ea;
            tour[0].wait := 0;
            tour[0].load := 0;

            {initialize vehicle nodes and terminal depot}
            FOR i := nc+1 TO numnodes
                tour[i] := CLONE(tour[0]);
                tour[i].id := i;
            END FOR;

    END METHOD; {readLatLongLoiter}

    {Compute 2 dimensional distance matrix given coords in same distance units
     as vehicle airspeed}
    {Wind NOT taken into account}
    {Does not assume the problem is symmetric, but makes it so}
    ASK METHOD distMatrix(IN nc , numnodes : INTEGER;
                          IN coord : coordArrType;
                          OUT dist : arrReal2dimType;
                          IN outstrm : StreamObj );
    VAR
        i, j,
        k : INTEGER;
        xdiff, ydiff,                       {differences for dist calc}
        xdiff2, ydiff2      : REAL;         {squared differences}
    BEGIN
    NEW(dist, 0..numnodes, 0..numnodes);    {initialize dist matrix}
```

```
{Find integer euclidean dist}
{depot and demand nodes}
FOR i := 0 TO nc-1
        FOR j := i+1 TO nc
                xdiff := coord[i].x - coord[j].x;
                ydiff := coord[i].y - coord[j].y;
                xdiff2 := xdiff*xdiff;
                ydiff2 := ydiff*ydiff;
                dist[i][j] := SQRT(xdiff2 + ydiff2);
                dist[j][i] := dist[i][j];
        END FOR;
END FOR;

{Ensure triangle inequality holds}
FOR i := 0 TO nc-1
        FOR j := i+1 TO nc
                FOR k := 0 TO nc
                 IF (k <> i) AND (k <> j)
                  IF dist[i][j] > dist[i][k] + dist[k][j]
                        dist[i][j] := dist[i][k] + dist [k][j];
                        dist[j][i] := dist[i][j];
                  END IF;
                 END IF;
                END FOR;
        END FOR;
END FOR;

{demand to vehicle & vehicle to demand travel same as demand to depot}
FOR i := 1 TO nc
        FOR j := nc+1 TO numnodes
                dist[i][j] := dist[0][i];
                dist[j][i] := dist[i][j];
        END FOR;
END FOR;

END METHOD; {distMatrix}

{Compute 2 dimensional distance matrix given Latitude and Longitude coords}
{Does not take wind into account}
{Does not assume the problem is symmetric, but makes it so}
ASK METHOD distLatLong(IN nc , numnodes : INTEGER;
                        IN coord : coordArrType;
                        OUT dist : arrReal2dimType;
                        IN startprint : BOOLEAN;
                        IN outstrm : StreamObj);
VAR
        i, j,
        k : INTEGER;
        str : STRING;
        angdiff   : REAL;          {angular difference in radians}
BEGIN
NEW(dist, 0..numnodes, 0..numnodes);        {initialize dist matrix}

IF startprint        ASK outstrm WriteLn;  END IF;
```

```
        {Find integer euclidean dist}
        {depot and demand nodes}
        FOR i := 0 TO nc
                FOR j := i+1 TO nc
                 angdiff := ACOS( SIN(coord[i].x) * SIN(coord[j].x)
                                        + COS(coord[i].x) * COS(coord[j].x)
                                          * COS( ABS(coord[j].y - coord[i].y) ) );

                 {57.2958 degrees per radian, 60 naut miles per degree}
                 dist[i][j] := 57.2958 *  60.0 * angdiff;
                 dist[j][i] := dist[i][j];

IF startprint
str := "i = "+INTTOSTR(i)+" j = "+INTTOSTR(j)+" dist = "+REALTOSTR(dist[i][j]);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;

                END FOR;
        END FOR;

        {Ensure triangle inequality holds}
        FOR i := 0 TO nc
                FOR j := i+1 TO nc
                        FOR k := 0 TO nc
                         IF (k <> i) AND (k <> j)
                          IF dist[i][j] > dist[i][k] + dist[k][j]
                                dist[i][j] := dist[i][k] + dist [k][j];
                                dist[j][i] := dist[i][j];
                          END IF;
                         END IF;
                        END FOR;
                END FOR;
        END FOR;

        {demand to vehicle & vehicle to demand travel same as demand to depot}
        FOR i := 1 TO nc
                FOR j := nc+1 TO numnodes
                        dist[i][j] := dist[0][i];
                        dist[j][i] := dist[i][j];
                END FOR;
        END FOR;

        END METHOD; {distLatLong}

        {Compute 2 dimensional time matrix}
        {Does take wind into account}
        {Does not assume the problem is symmetric, but makes it so}
        ASK METHOD timeMatrix(IN nc , numnodes,
                                gamma,          {cost placed on veh - veh arcs}
                                as,             {uav air speed}
                                wmag : INTEGER;         {magnitude of the wind vector}
                                IN wdir, {direction of wind vector}
                                  windconv : REAL;
                                IN coord : coordArrType;
                                IN s : arrIntType;
```

```
                           IN dist : arrReal2dimType;
                           OUT time : arrInt2dimType;
                           IN outstrm : StreamObj;
                           IN print : BOOLEAN);
VAR
        i, j, k
          : INTEGER;
        xdiff, ydiff,
        gs2,      {gs squared}
        gs,       {ground speed}
        Theta,    {angle between due EAST and ground speed vector}
        Phi       {angle between gs vector and wind vector}
          : REAL;
        str : STRING;
BEGIN

NEW(time, 0..numnodes, 0..numnodes);      {initialize time matrix}

{Find the angle from 0 (due EAST) to ground speed vector (gs)}
FOR i := 0 TO nc
  FOR j := 0 TO nc
    IF i <> j

        xdiff := coord[j].x - coord[i].x;
        ydiff := coord[j].y - coord[i].y;

        IF ABS(xdiff) < 0.00001

         IF ydiff > 0.0
           Theta := 90.0 * pi / 180.0;
         ELSE
           Theta := 270.0 * pi / 180.0;
         END IF;

        ELSE

         IF ABS(ydiff) < 0.00001

          IF xdiff > 0.0
           Theta := 0.0;
          ELSE
           Theta := pi; {180 degs}
          END IF;

         ELSIF ydiff > 0.0

          Theta := ACOS(xdiff / dist[i][j]);

         ELSE {ydiff < 0.0}

          Theta := 2.0 * pi - ACOS(xdiff / dist[i][j]);  {2*pi = 360 degs}

         END IF;
        END IF;
```

```
                Phi := wdir - Theta;
                gs2 := FLOAT(as*as) +FLOAT(wmag*wmag) - 2.0*FLOAT(as*wmag) *COS(Phi);
                gs := SQRT(gs2);

                time[i][j] := TRUNC( windconv * dist[i][j] / gs );
IF print
str := "i="+INTTOSTR(i)+" j="+INTTOSTR(j)+" Theta="+REALTOSTR(Theta)
        +" Phi="+REALTOSTR(Phi)+" cos(Phi)="+REALTOSTR(COS(Phi))
        +" GS="+REALTOSTR(gs)+" time="+INTTOSTR(time[i][j]);
ASK outstrm WriteString(str);  ASK outstrm WriteLn;
END IF;
            ELSE
              time[i][j] := 0;

            END IF; {i<>j}
          END FOR;
        END FOR;

        {Ensure triangle inequality holds}
        FOR i := 0 TO nc
                FOR j := 0 TO nc
                        FOR k := 0 TO nc
                          IF (k <> i) AND (k <> j)
                            IF time[i][j] > time[i][k] + time[k][j]
                                   time[i][j] := time[i][k] + time [k][j];
                            END IF;
                          END IF;
                        END FOR;
                END FOR;
        END FOR;

        {demand to vehicle & vehicle to demand travel same as demand to depot}
        FOR i := 1 TO nc
                FOR j := nc+1 TO numnodes
                        time[i][j] := time[0][i];
                        time[j][i] := time[i][j];
                END FOR;
        END FOR;

        {Add service time for all demand/demand and demand/vehicle arcs}
        FOR i := 0 TO nc
                FOR j := 0 TO numnodes
                        IF i <> j
                                time[i][j] := time[i][j] + s[i];
                        END IF;
                END FOR;
        END FOR;

        {Add vehicle usage penalty "gamma" to all vehicle to vehicle arcs}
        FOR i := nc+1 TO numnodes-1
                time[0][i] := time[0][i] + gamma;
                FOR j := i+1 TO numnodes
                        time[i][j] := time[i][j] + gamma;
                        time[j][i] := time[i][j];
                END FOR;
```

```
END FOR;

END METHOD; {timeMatrix}

END OBJECT; {timeMatrixObj}


OBJECT startUAVObj;

ASK METHOD startUAVbest (IN numnodes, tvl,      {travel time of tour}
                       tourLen : INTEGER;       {length of curr tour}
                  IN tour : tourType;   {curr tour}
                  IN TWPEN : REAL;              {mult factor for TW pen}
                  IN psurv : arrRealType;       {prob of survival array}
                  OUT totPenalty,               {total Penalty (TW here)}
                      penTrav,                  {tvl and twpen w/our wait}
                      tourCost          {tourLen + TW cost}
                              : INTEGER;
                  OUT tourPen : vrpPenType;{tour penalty record}
                  OUT bfiter,           {iter # of bf tour fd}
                       bfCost,          {lowest feas cost found}
                       bfTT,                    {best feas trav time}
                       bfnv,
                       bestiter,
                       bestCost,                {lowest cost found}
                       bestTT,
                       bestnv,
                   bfTime,              {Time best feas found}
                   bestTime             {Time best tour found}
                      : INTEGER;
                  OUT cvrg,
                   bfCvrg,
                   bestCvrg  : REAL;
                  OUT bestTour,                 {The best tour found}
                    bfTour  : tourType); {best feas tour}

VAR
        iter, nvu : INTEGER;
BEGIN

{initialize best FEASIBLE stuff}
{make the initial best penalties really large}

bfCost := 999999999;
bfTT := 9999999;
bfnv := 9999;
bfCvrg := 0.0;
bfTime := 0;
bfiter := -1;

NEW(bfTour, 0..numnodes);

bestCost := 999999999;
bestTT := 9999999;
bestnv := 9999;
```

```
        bestCvrg := 0.0;
        bestTime := 0;
        bestiter := -1;

        NEW(bestTour, 0..numnodes);

        {compute infeasibilities and costs}
        {note: if totPenalty > 0, tour NOT feasible}
        NEW(tourPen);               {Tour Penalty record iniialized}

        compPens(numnodes, tour, 0, tourPen);
        tsptwPen(numnodes, tourLen, tour, tourPen, TWPEN, totPenalty, tourCost,
                penTrav, tvl);

        countVeh(numnodes, tour, nvu);
        expCvrg(numnodes, psurv, tour, cvrg);

        uavBest(cvrg, numnodes, totPenalty, penTrav, tvl, nvu, 0, tour, bfCost,
               bfTT, bfnv, bfiter, bestCost, bestTT, bestnv, bestiter,
               bfCvrg, bestCvrg,
               bfTour, bestTour, bfTime, bestTime);

        END METHOD; {startPenBest}

END OBJECT {startUAVobj};


OBJECT uavRTSobj;

{Steps through ITER iterations of the UAV reactive tabu search.}
{The UAV search uses the max coverage procedure to evaluate move Value.
 The factor PSFCT converts the coverage value to integer.
 PSFCT is determined by the user as a function of the number of vehicles, and
 the number of targets.
 The TW constraints are still used to determine feasibility}
{The best tour is that of maximum coverage}

ASK METHOD search (IN psurv : arrRealType;             {Prob of survival array}
                  IN PSFCT : REAL;
                  IN TWPEN, INCREASE, DECREASE : REAL;
                  IN HTSIZE, CYMAX, ZRANGE, DEPTH, minTL, maxTL, tabuLen,
                        iters, nc, numnodes : INTEGER;
                  IN outstrm, outstrm2  : StreamObj;
                  INOUT tourPen : vrpPenType;
                  IN time : arrInt2dimType;
                  IN stepprint, moveprint, cycleprint : BOOLEAN;
                  INOUT tourCost, penTrav, totPenalty, tvl,
                        bfCost, bfTT, bfnv, bfiter, bestCost, bestTT, bestnv,
                        bestTime, bfTime, bestiter, numfeas : INTEGER;
                  INOUT bfCvrg, bestCvrg, cvrg : REAL;
                  INOUT tour, bestTour, bfTour : tourType);
VAR
        {UAV specific}
        nbrcvrg          {  "        "    "  "  tour being evaluated}
              : REAL;
```

```
travVal : INTEGER;  {move Value to the mTSPTW model}

{from original}
i,                          {index, usually current node for moving}
j,
k,                          {iteration number}
l,                          {index only}
fhv,                        {Woodruff&Zemel 1st level hash value}
shv,                        {Woodruff&Zemel 2nd level hash value}
tourLen,            {entire length of time tour takes}
ssltlc,             {steps since last tabu length change}
escBest, {the objective value of the best of all moves}
Dbest,              {smallest swap cost among all neighbors}
Dbestf,             {smallest swap cost among feasible neighbors}

escBestTT,          {trav time of the best neighbor of all moves}
DbestTT,            {trav time of smallest swap cost neighbor}
DbestfTT,           {trav time of smallest swap cost feasible neighbor}

chI,                {choice node initiating overall best insert move}
chD,                {choice node receiving overall best insert move}
           {"ch"'s may be initially set to nontabu infeasible moves
            or infeasible moves that aspire at insert move search  }
feasI,              {node initiating "good" feasible insert move}
feasD,              {node receiving "good" feasible insert move}
escI,               {node initiating "good" escape insert move}
escD,               {node receiving "good" escape insert move}
nodetype,           {type of node considered for insertion}
nexttype,           {type of node next to the considered insert node}
next2type,          {type of 2 steps from the considered insert node}
lf,                 {id of node on left}
rt,                 {id of node on right}
d,                  {index for insert DEPTH}
dstart,             {initial value for DEPTH index in EARLY loop}
moveVal,            {move value (curr tour to nbr), tvl + pen change}
totNbrPen,          {total penalty for neighbor tour}
zin,                {zin and zout update the tour hash value}
zout,               {   for the affected nodes only}
nvu                 {# vehicles used}
: INTEGER;

mavg    : REAL;             {moving average of cycle length}

tabulist : arrInt2dimType;

list : hashListObj;{used to instantiate the array of lists}
zArr : arrIntType;{random weights assigned to nodes}
node : nodeType;            {used to instantialte "working" tour}

load,
earlymove,                  {TRUE if an early move is to be performed}
found : BOOLEAN;            {TRUE if curr tour was visited before}

hashcurr : hashRecord;      {curr tour's 2nd hash and other info}
hashtbl : hashTblType;      {array of hash lists indexed by 1st hash}
```

```
        nbrtour : tourType;         {working tour for insertion operation}
        nbrPen : vrpPenType;        {penalty record of neighbor tour}
        str,
        where : STRING;             {used as diagnostic check}

CONST
{     iter  penTrav  bestCost   bfCost}
format="****< *******< ********< *******<";

BEGIN
{**
ASK outstrm2 WriteString("iter tabuLen penTrav bestCost bfCost");
ASK outstrm2 WriteLn;
**}

{initialize the RTS parameters}
mavg := FLOAT(numnodes - 2);
ssltlc := 0;

{initialize tabu array to zero}
NEW(tabulist, 0..numnodes, 0..numnodes);
FOR i := 0 TO numnodes
        FOR j := 0 TO numnodes
                tabulist[i][j] := 0;
        END FOR;
END FOR;

k := 1; {first iteration}
numfeas := 0;

NEW(hashtbl, 0..HTSIZE);                    {instantiate the hash table}
FOR i := 0 TO HTSIZE
        NEW(list);
        hashtbl[i] := list;
END FOR;

randWtWZ(nc, numnodes, ZRANGE, zArr);          {assign random wts to each node}

tourHVwz(numnodes, tour, zArr, shv);        {start tour's 2nd level hash val}

{Place initial tour in hash table}
{create a hashRecord}
NEW(hashcurr);

{assign the 2nd level hash value, tourCost, tvl & penalties hashcurr}
hashcurr.shv := shv;
hashcurr.cost := tourCost;
hashcurr.tvl := tvl;
hashcurr.twpen := tourPen.tw;
hashcurr.lastiter := 0;

{find the 1st level hash value for the current tour}
fhv := tourCost MOD HTSIZE;

{Add current hash record object to the linked list indexed by fhv}
```

```
ASK hashtbl[fhv] AddFirst(hashcurr);

{********** MAJOR SEARCH LOOP **************}
WHILE k <= iters

IF moveprint
ASK outstrm WriteLn;
str := "k = " + INTTOSTR(k);
ASK outstrm WriteString(str);
END IF;

  {** FIND INCUMBENT TOUR **}
  {initialize "travel value" parameters}
  DbestTT := 999999;
  escBestTT := 999999;
  DbestfTT := 999999;
  {initialize "move value" parameters}
  Dbest := 999;
  escBest := 999;
  Dbestf := 999;
  chI := 0;
  chD := 0;
  feasI := 0;
  feasD := 0;
  escI := 0;
  escD := 0;

  {******************************}
  {** check all LATER insertions**}
  FOR i := 1 TO numnodes-2

    {copy incumbent tour to working copy "nbr"}
    NEW(nbrtour, 0..numnodes);
    nbrtour := CLONE(tour);
    FOR l := 0 TO numnodes
          nbrtour[l] := CLONE(tour[l]);
    END FOR;

    nodetype := tour[i].type;          {determine current nodes type}

    d := 1;
    WHILE d <= DEPTH  {DEPTH loop}

          IF i+d < numnodes { -1 }{feasible depth?}

            {determine type of node on right}
            nexttype := tour[i+d].type;

            IF nodetype = 1 {customer node}

              {if strong TWs violated within a vehicle, move the customer
              along until a vehicle is encountered, then swap and
              "locally" update the schedule as the customer is
              swapped, and increment d as well}
```

```
{strong TW check}
IF (tour[i+d].ea + time[tour[i+d].id][tour[i].id])
   > tour[i].la

  WHILE nbrtour[i+d].type = 1

       lf := i+d-1;
       rt := i+d;
       SwapNode(nbrtour[lf], nbrtour[rt]);

       {local update:arr,dep,wait}
       nbrtour[lf].arr := nbrtour[lf-1].dep +
                          time[nbrtour[lf-1].id][nbrtour[lf].id];
       nbrtour[lf].dep := MAXOF(nbrtour[lf].ea, nbrtour[lf].arr);
       nbrtour[lf].wait :=nbrtour[lf].dep - nbrtour[lf].arr;

       {local update:load}
       IF nbrtour[lf-1].type = 2
        nbrtour[lf].load := nbrtour[lf].qty;
       ELSE
        nbrtour[lf].load := nbrtour[lf-1].load + nbrtour[lf].qty;
       END IF;

       d := d + 1;

       {IF with EXIT from "DEPTH" loop}
       {because if you increment to numnodes-1, don't want}
       {to do a swap with terminal depot}
       IF i+d = numnodes-1 EXIT; END IF;

 END WHILE;

END IF;{TW check}

{The customer is now ready to have its move evaluated:
 TRAVEL PORTION
 1 Swap it with the next node
 2 Compute the change in travel distance, and compute the
   neighbor's schedule
3 Compute the neighbor's penalty values
 4 Increase the total move value by the "costed penalties"}
{COVERAGE PORTION
 5 Find the neighbor's coverage
 6 Subtract nbr tour's cvrg from curr tour's coverage for moveVal}

{1} SwapNode(nbrtour[i+d-1], nbrtour[i+d]);
{2} moveValTT(i, d, numnodes, tour, nbrtour, time, travVal);
{3} NEW(nbrPen);
   compPens(numnodes, nbrtour, 0, nbrPen);
{4} totNbrPen := nbrPen.tw;
   travVal:= travVal+TRUNC(TWPEN*FLOAT(nbrPen.tw-tourPen.tw));

{5} expCvrg(numnodes, psurv, nbrtour, nbrcvrg);
{6} moveVal := TRUNC(PSFCT * (cvrg - nbrcvrg));
```

```
                DISPOSE(nbrPen);

IF stepprint
ASK outstrm WriteLn;
str := "node " + INTTOSTR(tour[i].id) + " d = " + INTTOSTR(d) + " ";
ASK outstrm WriteString(str);
str := " moveVal = " + INTTOSTR(moveVal) + " totNbrPen = " + INTTOSTR(totNbrPen);
ASK outstrm WriteString(str);
 IF i+d < numnodes
 IF k <= tabulist[tour[i].id][i+d]
        ASK outstrm WriteString(" **TABU");
 END IF;
 END IF;
END IF;

          {END nodetype = 1 (customer)}

         ELSE {nodetype = 2, vehicle
                and vehicles are always strong TW feasible
                IF next node is a customer, move is valid}

          IF nexttype = 2 EXIT; END IF;
          {dont swap adjacent vehicles, leave "d" loop}
          {1} SwapNode(nbrtour[i+d-1], nbrtour[i+d]);
          {2} moveValTT(i, d, numnodes, tour, nbrtour, time, travVal);

          {3} NEW(nbrPen);
             compPens(numnodes, nbrtour, 0, nbrPen);
          {4} totNbrPen := nbrPen.tw;
             travVal:= travVal+TRUNC(TWPEN*FLOAT(nbrPen.tw-tourPen.tw));

          {5} expCvrg(numnodes, psurv, nbrtour, nbrcvrg);
          {6} moveVal := TRUNC(PSFCT * (cvrg - nbrcvrg));

                DISPOSE(nbrPen);

IF stepprint
ASK outstrm WriteLn;
str := "node " + INTTOSTR(tour[i].id) + " d = " + INTTOSTR(d) + " ";
ASK outstrm WriteString(str);
str := " moveVal = " + INTTOSTR(moveVal) + " totNbrPen = " + INTTOSTR(totNbrPen);
ASK outstrm WriteString(str);
 IF i+d < numnodes
 IF k <= tabulist[tour[i].id][i+d]
        ASK outstrm WriteString(" **TABU");
 END IF;
 END IF;
END IF;

         END IF; {nodetype check}

         IF totNbrPen = 0             {feasible candidate tour?}

          {If this is best feasible neighbor,AND not tabu OR it aspires, SAVE}
          IF (moveVal < Dbestf)
```

```
        OR ( (moveVal = Dbestf) AND (travVal < DbestfTT) )

    IF ( k > tabulist[tour[i].id][i+d] )
     OR ( nbrcvrg < bestCvrg )

      Dbestf := moveVal;
      feasI := i;
      feasD := d;
      DbestfTT := travVal;

    END IF; {not tabu OR aspires}
  END IF; {moveVal < DbestF}
 {END totNbrPen = 0}

 ELSE {candidate is infeasible}

  {IF this is best infeas neighbor, SAVE}
  IF (moveVal < Dbest)
        OR ( (moveVal = Dbest) AND (travVal < DbestTT) )

        IF ( k > tabulist[tour[i].id][i+d] )
         OR ( nbrcvrg < bestCvrg )

         Dbest := moveVal;
         chI := i;
         chD := d;
         DbestTT := travVal;

        END IF; {not tabu OR aspires}
  END IF; {moveVal < Dbest}
 END IF; {infeas candidate}

 {Escape Routine}
 {saves the best of all neighbor moves in case all moves tabu or
 non-quality changing}
 IF (moveVal < escBest)
   OR ( (moveVal = escBest) AND (travVal < escBestTT) )

   escBest := moveVal;
   escI := i;
   escD := d;
   escBestTT := travVal;

 END IF;{escape}

{IF only vehicle nodes are left in the tour, STOP, }
{get the next node. Compare the position to the id of the }
{node, IF equal you are at the end of the tour (Carlton, 95:5.3)}
 IF ( nbrtour[i+d+1].type = 2 )
  AND ( nbrtour[i+d+1].id = i + d + 1 )

  EXIT;  END IF;

ELSE   {i+d < numnodes - 1 (feasible DEPTH)}
```

```
            EXIT;

          END IF;

          d := d + 1;

     END WHILE;              {d = 1 to DEPTH}

     FOR l := 0 TO numnodes
          DISPOSE(nbrtour[l]);
     END FOR;
     DISPOSE(nbrtour);

IF stepprint
ASK outstrm WriteLn;
str := "Dbestf = " + INTTOSTR(Dbestf) + " Dbest = " + INTTOSTR(Dbest)
      + " escBest = " + INTTOSTR(escBest);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;

   END FOR;{i = 1 to numnodes-2}

   {***********************************}
   {*** check all EARLIER insertions ***}

   i := 3

   WHILE i <= numnodes-1

     earlymove := TRUE; {initially, we intend to perform a move}

     {create working copy}
     nbrtour := CLONE(tour);
     FOR l := 0 TO numnodes
          nbrtour[l] := CLONE(tour[l]);
     END FOR;

     {do not consider any d = -1 moves as they are later moves}
     d := 1;

     nodetype := tour[i].type;
     nexttype := tour[i-1].type;
     next2type := tour[i-2].type;

     IF nodetype = 2 {vehicle node}

       IF (nexttype <> 2) AND (next2type <> 2)
       {dont want adjacent vehicles or a sandwiched customer}

            SwapNode(nbrtour[i-d], nbrtour[i-d+1]);
            d := d + 1;

       ELSE

            earlymove := FALSE; {GOTO NEXT NODE}
```

```
    END IF; {nexttype or next2type = 2}

ELSE {customer node}

 {strong TW check}
 IF tour[i].ea + time[tour[i].id][tour[i-1].id] <= tour[i-1].la
  {do the d = -1 swap (i and i-1)}

  SwapNode(nbrtour[i-d], nbrtour[i-d+1]);
  d := d + 1;

    ELSE {TW check NOT OK}

  {do swaps to the next earlier vehicle node}
  {stop while a customer is adjacent}
  WHILE nbrtour[i-d].type = 1
   SwapNode(nbrtour[i-d], nbrtour[i-d+1]);
   d := d + 1;

  END WHILE;

  {if we are now at start depot, GOTO NEXT NODE}
  IF i-d = 0
   earlymove := FALSE;
  END IF;

 END IF; {strong TW check}

END IF; {END for customer node}

IF earlymove = TRUE

 WHILE d <= DEPTH          {DEPTH loop}

  IF i-d <= 0 {feasible DEPTH check}

   EXIT;    {avoid unnecessary loops}

  ELSE

   IF nodetype = 1
    {strong TW check}
    IF tour[i].ea + time[tour[i].id][tour[i-d].id]
      > tour[i-d].la

     {swap adjacent customers}
     WHILE nbrtour[i-d].type = 1

     SwapNode(nbrtour[i-d], nbrtour[i-d+1]);
     d := d + 1;
     END WHILE;

     {stop at node 0, GOTO NEXT NODE (i)}
     IF i-d = 0
```

```
EXIT;
END IF;

END IF; {strong TW check}

{*now evaluate neighbor tour*}

   {1} SwapNode(nbrtour[i-d], nbrtour[i-d+1]);
   {2} moveValTT(i, -d, numnodes, tour, nbrtour, time, travVal);
   {3} NEW(nbrPen);
     compPens(numnodes, nbrtour, 0, nbrPen);
   {4} totNbrPen := nbrPen.tw;
     travVal:= travVal+TRUNC(TWPEN*FLOAT(nbrPen.tw-tourPen.tw));

   {5} expCvrg(numnodes, psurv, nbrtour, nbrcvrg);
   {6} moveVal := TRUNC(PSFCT * (cvrg - nbrcvrg));

        DISPOSE(nbrPen);

IF stepprint
ASK outstrm WriteLn;
str := "node " + INTTOSTR(tour[i].id) + " d = " + INTTOSTR(-d) + " ";
ASK outstrm WriteString(str);
str := " moveVal = " + INTTOSTR(moveVal) + " ";
ASK outstrm WriteString(str);
 IF i+d < numnodes
 IF k <= tabulist[tour[i].id][i+d]
       ASK outstrm WriteString(" **TABU");
 END IF;
 END IF;
END IF;

     ELSE      {END for customer node, start vehicle node}

      nexttype := tour[i-d-1].type;

      {dont swap to adjacent vehicles, eval next node}
      IF (nexttype = 2)

              EXIT; {GOTO NEXT NODE (i)}

      ELSE

       {*evaluate neighbor tour*}
          {1} SwapNode(nbrtour[i-d], nbrtour[i-d+1]);

          {2} moveValTT(i, -d, numnodes, tour, nbrtour, time, travVal);
          {3} NEW(nbrPen);
            compPens(numnodes, nbrtour, 0, nbrPen);
          {4} totNbrPen := nbrPen.tw;
            travVal:= travVal+TRUNC(TWPEN*FLOAT(nbrPen.tw-tourPen.tw));

          {5} expCvrg(numnodes, psurv, nbrtour, nbrcvrg);
          {6} moveVal := TRUNC(PSFCT * (cvrg - nbrcvrg));
```

```
                DISPOSE(nbrPen);

IF stepprint
ASK outstrm WriteLn;
str := "node " + INTTOSTR(tour[i].id) + " d = " + INTTOSTR(-d) + " ";
ASK outstrm WriteString(str);
str := " moveVal = " + INTTOSTR(moveVal) + " ";
ASK outstrm WriteString(str);
 IF i+d < numnodes
 IF k <= tabulist[tour[i].id][i+d]
       ASK outstrm WriteString(" **TABU");
 END IF;
 END IF;
END IF;

          END IF; {END adjacent vehicle check}

      END IF; {END for vehicle node}

      {feasible tour?}
      IF totNbrPen = 0

       IF (moveVal < Dbestf)
            OR ( (moveVal = Dbestf) AND (travVal < DbestfTT) )

        {IF not tabu OR aspires}
        IF ( k > tabulist[tour[i].id][i-d] )
         OR ( nbrcvrg < bestCvrg )

         Dbestf := moveVal;
         feasI := i;
         feasD := -d;
         DbestfTT := travVal;

        END IF; {IF not tabu OR aspires}
       END IF; {moveVal < Dbestf}

      ELSE {infeasible tour}

       IF (moveVal < Dbest)
            OR ( (moveVal = Dbest) AND (travVal < DbestTT) )

        {IF not tabu OR aspires}
        IF ( k > tabulist[tour[i].id][i-d] )
         OR ( nbrcvrg < bestCvrg )

         Dbest := moveVal
         chI := i;
         chD := -d;
         DbestTT := travVal;

        END IF; {IF not tabu OR aspires}
       END IF; {moveVal < Dbest}

      END IF; {feasible tour check}
```

```
          {Escape Routine}
          {saves the best of all neighbor moves in case all moves tabu
           or non-quality changing}
          IF (moveVal < escBest)
           OR ( (moveVal = escBest) AND (travVal < escBestTT) )
           escBest := moveVal
           escI := i;
           escD := -d;
           escBestTT := travVal;

          END IF;{escape}

      END IF; {feasible DEPTH check}

          d := d + 1;

    END WHILE; {DEPTH loop}

   END IF; {earlymove=TRUE}

   i := i + 1;

IF stepprint
ASK outstrm WriteLn;
str := "Dbestf = " + INTTOSTR(Dbestf) + " Dbest = " + INTTOSTR(Dbest)
      + " escBest = " + INTTOSTR(escBest);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;

   FOR l := 0 TO numnodes
         DISPOSE(nbrtour[l]);
   END FOR;
   DISPOSE(nbrtour);

  END WHILE; {i = 3 TO numnodes-1}

  {If feasible move found, move to it}
  IF feasI <> 0
   chI := feasI;
   chD := feasD;


  {** IF ALL MOVES ARE TABU AND NONE MEET ASPIRATION CRITERIA **}
  {                                                           }
  {  THEN SET chI AND chD TO THE BEST MOVE DISCOVERED                }
  {     AND DECREASE THE TABU LENGTH                          }
  {                                                           }
  {        OR IF NO MOVES ARE AVAILABLE                       }
  {************************************************************}

  {NO MOVES ARE AVAILABLE}
  {This "degenerate" condition only occurs whenever only one
   vehicle is available and no feasible moves are available
   because of STRONG TW feasibility.  This stops all computation
```

```
and prompts the user to restart allowing more than one vehicle.}

ELSIF escI = 0
  ASK outstrm WriteLn;
  ASK outstrm WriteInt(k, 4);
  ASK outstrm WriteString("There are no moves available....");
  ASK outstrm WriteString("Increase the number of vehicles and try again");
  ASK outstrm WriteLn;
  EXIT;

{ALL MOVES ARE TABU AND NONE MEET ASPIRATION CRITERIA}
ELSIF chI = 0

ASK outstrm WriteString("All moves tabu and none meet aspiration criteria ");
ASK outstrm WriteString("at iteration: ");
ASK outstrm WriteInt(k, 4);
ASK outstrm WriteLn;

  {best of the neighbors is still moved to, tabu length adjusted}
  chI := escI;
  chD := escD;
  tabuLen := MAXOF( ROUND(FLOAT(tabuLen) * DECREASE), minTL);
END IF;

{** UPDATE TABU LIST AND TOUR POSITIONS **}
{allow no "return" moves for tabuLen iterations, See Carlton '95: }
{4.3.6. Prevents a direct (active) move back to the position }
{which the node just moved from}

IF chD = 1
  tabulist[tour[chI+1].id][chI+1] := k + tabuLen;
ELSE
  tabulist[tour[chI].id][chI] := k + tabuLen;
END IF;

{allow no "repeat" moves for tabuLen iterations, See Carlton '95: }
{4.3.6.  Prevents a direct (active) move back into the position }
{into which the node is currently moving}

tabulist[tour[chI].id][chI+chD] := k + tabuLen;

{BEFORE the new tour is constructed, update the tour hashing value}
{Performed exactly like a 3-opt move update, Wooruff&Zemel (93)}
zin := 0;  zout := 0;

i := chI;
IF chD > 0
  j:= chI + chD;
ELSE
  j:= chI + chD - 1;
END IF;

zout := (zArr[tour[i-1].id] * zArr[tour[i].id])
     + (zArr[tour[i].id] * zArr[tour[i+1].id])
     + (zArr[tour[j].id] * zArr[tour[j+1].id]);
```

```
  zin := (zArr[tour[i-1].id] * zArr[tour[i+1].id])
       + (zArr[tour[j].id] * zArr[tour[i].id])
       + (zArr[tour[i].id] * zArr[tour[j+1].id]);

  shv := shv + (zin - zout);

IF moveprint
ASK outstrm WriteLn;
str := "Move inserts node " + INTTOSTR(tour[chI].id) + " to position "
     + INTTOSTR(chI + chD);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
str := "w/ shv = " + INTTOSTR(shv);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;

  {Perform the insertion}
  insert(chI, chD, tour);

IF moveprint
qcktourFile(outstrm, tour, numnodes);
END IF;

  {*UPDATE THE NEW INCUMBENT SCHEDULE*}
  {* schedule data and tour length   *}
  IF chD > 0
        tourSched(chI, nc, numnodes, tour, time, tourLen, outstrm);
  ELSE
        tourSched(chI+chD, nc, numnodes, tour, time, tourLen, outstrm);
  END IF;

  {update penalties}
  compPens(numnodes, tour, 0, tourPen);
  tsptwPen(numnodes, tourLen, tour, tourPen, TWPEN, totPenalty,
           tourCost, penTrav, tvl);

  {UPDATE COVERAGE}
  expCvrg(numnodes, psurv, tour, cvrg);

IF moveprint
str := " and Tour Cost = " + INTTOSTR(tourCost);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
str := "Current mavg is " + REALTOSTR(mavg) + " and Steps since last TL change "
       + INTTOSTR(ssltlc) + " current tabuLen " + INTTOSTR(tabuLen);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;

 {*********}
 {CYCLE CHECK}

  fhv := tourCost MOD HTSIZE;
  lookfor(fhv, tourCost, shv, tvl, k, tourPen, hashtbl, hashcurr, found);

  {if exact match exists then we found a cycle}
```

```
  IF found = FALSE {new unfound feasible tour}
    IF totPenalty = 0
          numfeas := numfeas +1;
    END IF;

    countVeh(numnodes, tour, nvu);
    uavBest(cvrg, numnodes, totPenalty, penTrav, tvl, nvu, k, tour, bfCost,
             bfTT, bfnv, bfiter, bestCost, bestTT, bestnv, bestiter,
             bfCvrg, bestCvrg,
             bfTour, bestTour, bfTime, bestTime);
    nocycle(DECREASE, minTL, mavg, ssltlc, tabuLen, outstrm, cycleprint);

IF moveprint
str := "This tour was NOT FOUND in the hashing structure";
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;

  ELSE
    {use hashcurr to get correct "lastiter"}
    cycle(hashcurr, INCREASE, maxTL, CYMAX, k, mavg, ssltlc, tabuLen,
            outstrm, cycleprint);

IF moveprint
str := "This tour was FOUND in the hashing structure";
ASK outstrm WriteString(str); ASK outstrm WriteLn;
END IF;

  END IF;

{**
IF moveprint
str := " "
twLoadToFile(str, outstrm, tour, nc, numnodes, tourLen, TRUE);
END IF;
**}

{*** OUTPUT("k = ", k," and bestCost = ", bestCost); ***}
{**
IF (k MOD 10) = 0
 ASK outstrm2 WriteInt(k, 4); ASK outstrm WriteString(" ");
 ASK outstrm2 WriteInt(tabuLen, 4); ASK outstrm WriteString(" ");
 ASK outstrm2 WriteInt(penTrav, 7); ASK outstrm WriteString(" ");
 ASK outstrm2 WriteInt(bestCost, 7); ASK outstrm WriteString(" ");
 ASK outstrm2 WriteInt(bestCost, 7); ASK outstrm WriteString(" ");
 ASK outstrm2 WriteLn;
END IF;
**}
  k := k + 1;

END WHILE; {* TABU SEARCH ROUTINE END *}

DISPOSE(hashtbl);
DISPOSE(tabulist);
DISPOSE(zArr);
```

```
END METHOD; {search}

END OBJECT; {uavRTSobj}


{Find the expected coverage for an entire tour}
{Total coverage is the sum of each vehicle's coverage}
{Each vehicle's coverage is the product of the probabilities of survival for
 the targets that vehicle visits}
PROCEDURE expCvrg(IN numnodes : INTEGER;
                  IN psurv : arrRealType;
                  IN tour : tourType;
                  OUT cvrg : REAL);       {coverage of tour}
VAR
        i,                          {index of tour array}
        j : INTEGER;                {index of the vehicle tours}
        vehcvrg  : REAL; {coverage performed by one vehicle}
        nodecvrg : arrRealType; {array holding the exp cvrg at each target}
BEGIN
        {Instantiate a temporary array to hold the}
        {expected coverage for each node visited by}
        {this vehicle.}
        NEW(nodecvrg, 1..numnodes)

        cvrg := 0.0;
        i := 0;

        WHILE i < numnodes

         {vehicle to demand transition starts veh tour}
         IF (tour[i].type = 2) AND (tour[i+1].type = 1)

          j := 1;           {start of the vehicle tour}
          vehcvrg := 0.0; {reset vehicle coverage}

          {find coverage of this vehicle}
          WHILE tour[i+j].type = 1

           IF j = 1
                nodecvrg[i+j] := psurv[tour[i+j].id];
           ELSE
                nodecvrg[i+j] := nodecvrg[i+j-1] * psurv[tour[i+j].id];
           END IF;

           {add the expected node cvrg to vehcvrg}
           vehcvrg := vehcvrg + nodecvrg[i+j]
           j := j + 1;

          END WHILE; {vehicle}

          {add vehicle cvrg to total cvrg}
          cvrg := cvrg + vehcvrg;

          {move past this veh tour to look for another veh tour}
          i := i + j;
```

```
ELSE
 IF i = numnodes - 1
  EXIT;
 ELSE
  i := i + 1;
 END IF;

END IF;

END WHILE;

DISPOSE(nodecvrg);

END PROCEDURE; {expCvrg}

{Print tour with random Service info to outstrm file}
PROCEDURE twServToFile(IN where : STRING;
                       IN outstrm :StreamObj;
                       IN tour : tourType;
                       IN nc, numnodes,
                       tourLen :INTEGER;
                       IN factor : REAL;
                       IN load : BOOLEAN;
                       IN s, slo, shi : arrIntType);
CONST
 {    id  eArr   lArr   | Arr     Dep     Wait | s     slo    shi  | Qty Load}
 format1="***< ****.*< ****.*<  *****.*< *****.*< ***.*< *.***< *.***< *.***< **< ***<";
 format2="***< ****.*< ****.*<  *****.*< *****.*< ***.*< *.***< *.***< *.***<";
VAR
      i : INTEGER;
      name, str : STRING;
      serv, servlo, servhi : REAL;
BEGIN
ASK outstrm WriteString(where);
ASK outstrm WriteLn;
ASK outstrm WriteString("Tour Length: ");
ASK outstrm WriteInt(tourLen,4);
ASK outstrm WriteLn;
ASK outstrm WriteString("Node information follows:");
ASK outstrm WriteLn;
IF load
 ASK outstrm WriteString("TYPE ID  eArr   lArr  | Arr    Dep     Wait | s");
 ASK outstrm WriteString("   slo  shi | Qty Load");
ELSE
 ASK outstrm WriteString("TYPE ID  eArr   lArr  | Arr    Dep     Wait | s");
 ASK outstrm WriteString("   slo  shi");
END IF;
ASK outstrm WriteLn;

FOR i := 0 TO numnodes
   IF ((tour[i].id = 0) OR (tour[i].id = numnodes))
     AND (tour[i].type = 2)
    name := "DEPOT ";
    serv := FLOAT(s[0]);
```

```
    servlo := FLOAT(slo[0]);
    servhi := FLOAT(shi[0]);
  ELSIF tour[i].type = 2
    name := "VHCL  ";
    serv := FLOAT(s[0]);
    servlo := FLOAT(slo[0]);
    servhi := FLOAT(shi[0]);
  ELSIF tour[i].type = 1
    name := "NODE  ";
    serv := FLOAT(s[tour[i].id]);
    servlo := FLOAT(slo[tour[i].id]);
    servhi := FLOAT(shi[tour[i].id]);
  END IF;

  IF load = TRUE
    str := SPRINT(tour[i].id, FLOAT(tour[i].ea) / factor,
          FLOAT(tour[i].la) / factor, FLOAT(tour[i].arr) / factor,
          FLOAT(tour[i].dep) / factor, FLOAT(tour[i].wait) / factor,
          serv / factor, servlo / factor, servhi / factor,
           tour[i].qty, tour[i].load )
        WITH format1;
  ELSE
    str := SPRINT(tour[i].id, FLOAT(tour[i].ea) / factor,
           FLOAT(tour[i].la) / factor, FLOAT(tour[i].arr) / factor,
           FLOAT(tour[i].dep) / factor, FLOAT(tour[i].wait) / factor,
           serv / factor, servlo / factor, servhi / factor)
        WITH format2;
  END IF;

  ASK outstrm WriteString(name);
  ASK outstrm WriteString(str);
  ASK outstrm WriteLn;
END FOR;

ASK outstrm WriteLn;

END PROCEDURE; {twServToFile}

{Print tour with coverage and service info to outstrm file}
PROCEDURE twCvrgServToFile(IN where : STRING;
                           IN outstrm :StreamObj;
                           IN tour : tourType;
                           IN nc, numnodes,
                           tourLen :INTEGER;
                           IN factor : REAL;
                           IN load : BOOLEAN;
                           IN psurv : arrRealType;
                           IN s, slo : arrIntType);
CONST
 {      id  eArr   lArr   l Arr     Dep     Wait l Ps   s    slo l Qty Load}
 format1="***< ****.*< ****.*<  *****.*< *****.*< ***.*< *.** ***.** ***.**<  **< ***<";
 format2="***< ****.*< ****.*<  *****.*< *****.*< ***.*< *.** ***.** ***.**<";
VAR
        i : INTEGER;
        name, str : STRING;
```

```
        ps, serv, servlo : REAL;
BEGIN
ASK outstrm WriteString(where);
ASK outstrm WriteLn;
ASK outstrm WriteString("Tour Length: ");
ASK outstrm WriteInt(tourLen,4);
ASK outstrm WriteLn;
ASK outstrm WriteString("Node information follows:");
ASK outstrm WriteLn;
IF load
 ASK outstrm WriteString("TYPE  ID  eArr   lArr   | Arr    Dep    Wait | Ps");
 ASK outstrm WriteString("   s  slo | Qty Load");
ELSE
 ASK outstrm WriteString("TYPE  ID  eArr   lArr   | Arr    Dep    Wait | Ps");
 ASK outstrm WriteString("   s  slo");
END IF;
ASK outstrm WriteLn;

FOR i := 0 TO numnodes
   IF ((tour[i].id = 0) OR (tour[i].id = numnodes))
     AND (tour[i].type = 2)
    name := "DEPOT ";
    ps := psurv[0];
    serv := FLOAT(s[tour[0].id]);
    servlo := FLOAT(slo[tour[0].id]);
   ELSIF tour[i].type = 2
    name := "VHCL  ";
    ps := psurv[0];
    serv := FLOAT(s[tour[0].id]);
    servlo := FLOAT(slo[tour[0].id]);
   ELSIF tour[i].type = 1
    name := "NODE  ";
    ps := psurv[tour[i].id];
    serv := FLOAT(s[tour[i].id]);
    servlo := FLOAT(slo[tour[i].id]);
   END IF;

   IF load = TRUE
    str := SPRINT(tour[i].id, FLOAT(tour[i].ea) / factor,
         FLOAT(tour[i].la) / factor, FLOAT(tour[i].arr) / factor,
         FLOAT(tour[i].dep) / factor, FLOAT(tour[i].wait) / factor, ps,
         serv / factor, servlo / factor,
         tour[i].qty, tour[i].load )
       WITH format1;
   ELSE
    str := SPRINT(tour[i].id, FLOAT(tour[i].ea) / factor,
          FLOAT(tour[i].la) / factor, FLOAT(tour[i].arr) / factor,
          FLOAT(tour[i].dep) / factor, FLOAT(tour[i].wait) / factor, ps,
          serv / factor, servlo / factor)
       WITH format2;
   END IF;

   ASK outstrm WriteString(name);
   ASK outstrm WriteString(str);
   ASK outstrm WriteLn;
```

```
END FOR;

ASK outstrm WriteLn;

END PROCEDURE; {twCvrgServToFile}

{Print tour with coverage info to outstrm file}
PROCEDURE twCvrgToFile(IN where : STRING;
                        IN outstrm :StreamObj;
                        IN tour : tourType;
                        IN nc, numnodes,
                        tourLen :INTEGER;
                        IN factor : REAL;
                        IN load : BOOLEAN;
                        IN psurv : arrRealType);
CONST
 {      id  eArr   lArr   | Arr     Dep     Wait | Ps   | Qty Load}
 format1="***< ****.*< ****.*<  *****.*< *****.*< ***.*< *.**<  **< ***<";
 format2="***< ****.*< ****.*<  *****.*< *****.*< ***.*< *.**<";
VAR
     i : INTEGER;
     name, str : STRING;
     ps : REAL;
BEGIN
ASK outstrm WriteString(where);
ASK outstrm WriteLn;
ASK outstrm WriteString("Tour Length: ");
ASK outstrm WriteInt(tourLen,4);
ASK outstrm WriteLn;
ASK outstrm WriteString("Node information follows:");
ASK outstrm WriteLn;
IF load
 ASK outstrm WriteString("TYPE ID  eArr   lArr   | Arr    Dep    Wait | Ps");
 ASK outstrm WriteString("| Qty Load");
ELSE
 ASK outstrm WriteString("TYPE ID  eArr   lArr   | Arr    Dep    Wait | Ps");
END IF;
ASK outstrm WriteLn;

FOR i := 0 TO numnodes
  IF ((tour[i].id = 0) OR (tour[i].id = numnodes))
    AND (tour[i].type = 2)
   name := "DEPOT ";
   ps := psurv[0];
  ELSIF tour[i].type = 2
   name := "VHCL  ";
   ps := psurv[0];
  ELSIF tour[i].type = 1
   name := "NODE  ";
   ps := psurv[tour[i].id];
  END IF;

  IF load = TRUE
   str := SPRINT(tour[i].id, FLOAT(tour[i].ea) / factor,
      FLOAT(tour[i].la) / factor, FLOAT(tour[i].arr) / factor,
```

```
            FLOAT(tour[i].dep) / factor, FLOAT(tour[i].wait) / factor, ps,
            tour[i].qty, tour[i].load )
          WITH format1;
      ELSE
        str := SPRINT(tour[i].id, FLOAT(tour[i].ea) / factor,
            FLOAT(tour[i].la) / factor, FLOAT(tour[i].arr) / factor,
            FLOAT(tour[i].dep) / factor, FLOAT(tour[i].wait) / factor, ps)
          WITH format2;
      END IF;

      ASK outstrm WriteString(name);
      ASK outstrm WriteString(str);
      ASK outstrm WriteLn;
    END FOR;

    ASK outstrm WriteLn;

    END PROCEDURE; {twCvrgToFile}



{best coverage, lowest completion time}
{Retains the feasible solution having the greatest coverage and with the
 greatest coverage has the lowest completion time
        first saves tour with greatest coverage
        ties broken by shortest completion time }
PROCEDURE uavBest (IN expCvrg : REAL;                   {expected coverage}
                   IN numnodes,
                      totPenalty,             {total penalty}
                      penTrav,                {current tour: penalty + tvl}
                      tvl,                    {current tour: travel time}
                      nvu,                    {number of vehicles used}
                      iter                    {current iteration number}
                          : INTEGER;
                    IN tour : tourType;     {current tour}
                    INOUT bfCost, bfTT, {best feas cost & tvl time}
                          bfnv,         {best feas num vehs used}
                          bfiter,       {iter # when best feas found}
                          bestCost,             {best overall penalty + TT}
                          bestTT,               {best overall travel time}
                          bestnv,               {best number of vehs used}
                          bestiter              {iter # when best ovrall found}
                          : INTEGER;
                    INOUT bfCvrg,               {best feas expected cvrg}
                          bestCvrg              {best overall exp cvrg}
                          : REAL;
                    INOUT bfTour, bestTour : tourType;
                    INOUT bfTime, bestTime : INTEGER );

VAR
        i,
        currtime   : INTEGER;     {current clock time of search}
BEGIN
        currtime := SystemTime();
```

```
{save the tour if it is the best ever found}
IF  expCvrg > bestCvrg

  bestCvrg := expCvrg;

  bestTT := tvl;
  bestCost := penTrav;
  bestTime := currtime;
  bestiter := iter;
  FOR i := 0 TO numnodes
          bestTour[i] := CLONE(tour[i]);
  END FOR;
  bestnv := nvu;

ELSIF (expCvrg = bestCvrg) AND (penTrav < bestCost)

  bestCost := penTrav;
  bestTT := tvl;
  bestTime := currtime;
  bestiter := iter;
  FOR i := 0 TO numnodes
          bestTour[i] := CLONE(tour[i]);
  END FOR;
  bestnv := nvu;

END IF;

{feasible checks}
IF (expCvrg < bfCvrg) OR (totPenalty > 0)
          RETURN;
ELSIF (expCvrg > bfCvrg) AND (totPenalty = 0)

          bfCvrg := expCvrg;

          bfTime := currtime;
          bfCost := penTrav;
          bfTT := tvl;
          bfiter := iter;
          FOR i := 0 TO numnodes
            bfTour[i] := CLONE(tour[i]);
          END FOR;
          bfnv := nvu;
          RETURN;

ELSIF (expCvrg = bfCvrg) AND (penTrav < bfCost)

          bfTime := currtime;
          bfCost := penTrav;
          bfTT := tvl;
          bfiter := iter;
          FOR i := 0 TO numnodes
            bfTour[i] := CLONE(tour[i]);
          END FOR;
          bfnv := nvu;
          RETURN;
```

```
        END IF;

        RETURN;

END PROCEDURE; {twbestTT}

END MODULE. {Implementation uavMod}
```

Appendix G: MuavLoiter

The main module MuavLoiter runs the *initialization phase* of UAV problems with stochastic winds and service times. The operator can adjust the random seeds, the airspeed, the wind parameters, and choose to read-in an initial tour. It creates the route frequency matrix and finds the "robust" tour after the specified number of days.

```
MAIN MODULE uavLoiter;

FROM IOMod IMPORT StreamObj, ALL FileUseType, ReadKey;
FROM OSMod IMPORT SystemTime;
FROM MathMod IMPORT pi;

FROM uavMod IMPORT timeMatrixObj;
FROM twReduceMod IMPORT twReductionObj;
FROM tsptwMod IMPORT startTourObj;
FROM tsptwMod IMPORT reacTabuObj;

FROM tabuMod IMPORT arrReal2dimType;
FROM tabuMod IMPORT coordArrType;
FROM tabuMod IMPORT tourType;
FROM tabuMod IMPORT nodeType;
FROM tabuMod IMPORT vrpPenType;
FROM tabuMod IMPORT arrInt2dimType;
FROM tabuMod IMPORT arrIntType;
FROM tabuMod IMPORT arrRealType;

FROM tabuMod IMPORT SwapNode;
FROM uavMod IMPORT twServToFile;
FROM tabuMod IMPORT LatLongToFile;
FROM tabuMod IMPORT qcktourFile;
FROM tabuMod IMPORT timeToFile;
FROM tabuMod IMPORT tourSched;
FROM tabuMod IMPORT countVeh;

FROM RandMod IMPORT RandomObj, SetSeed, FetchSeed;

VAR
        timeMatrix : timeMatrixObj;
        twReduce : twReductionObj;
        startTour : startTourObj;
        rts : reacTabuObj;
        randObj1, randObj2, randObj3, randObj4 : RandomObj;

        instrm,
        instrm2,
        outstrm,
        outstrm2,
```

```
outInit : StreamObj;

factor,    {used to convert the time windows to integer values}
TWPEN,           {Penalty weight assigned to the sum of late arr TW violations}
INCREASE,        {RTS parameter: mult. factor to decrease tabu length}
DECREASE,        {RTS parameter: mult. factor to increase tabu length}

windconv,   {multiplied by the resulting UAV time matrix, it provides an
            integer matrix (for calc speed) with the needed precision}
sumTij,          {sum of the i to j distances in the distance matrix}
mindist, {minimum travel distance}
maxdist, {maximum travel distance}
distAvg, {avg travel distance}
wdir,            {direction of wind vector}
ploiter,  {probabilitiy you loiter over a target}
loiter,          {loiter? - individual node result}
servSum          {sum of increase in service times}
        : REAL;

i, j, k,
endnum,          {end number in a numbered data file group}
maxtime,         {max possible time of arrival to any node, for time read}
numcycles,       {number of TW reduction cycles wanted}
numchanges,      {number of TWs reduced by TW reduction Obj}
numnodes,        {number of nodes in the directed graph}
nv,              {number of vehicles}
nc,              {number of targets/customers}
gamma,           {arbitrary cost assigned to the use of each vehicle}
iters,           {number of Tabu Search Iterations per problem}
tourLen, {Length of tour in time}
tvl,             {travel time of tour}
totPenalty,      {Total Penalty assigned to current tour}
tourCost,        {tour Length + Time Window Cost}
penTrav,         {tourCost - totWait == travel time + TW penalty}
bfCost,          {lowest tourCost found for a feasible tour}
bestCost,        {lowest tourCost found for a any tour}
bestTT,          {lowest travel time found for a any tour}
bestnv,          {# vehs used by best overall tour}
bfTT,            {lowest travel time found for feasible tour}
bfnv,            {# vehs used by best feas tour}
bfiter,    {iteration # when best feasible tour found}
tourhv,          {tour's hashing value}
bestiter, {iteration the best Tour found}
bestTime,        {Time the best Tour found}
bestTimeF,       {Time the best feasible Tour found}
numfeas,         {number of feasible solns found}
startTime,       {start Time (after time matrix, before TW reductions)}
stopTime,        {stop Time (after last iteration)}

DEPTH,           {depth of nodes we look for insert moves}
ZRANGE,                  {upper bound on random integer weights assigned to nodes}
HTSIZE,                  {size of hash table array}
CYMAX,                   {max cyleLength used to alter mavg}
tabuLen, {current length of tabu tenure}
minTL,           {minimum Tabu Length}
```

```
maxTL,          {maximum Tabu Length}

wmag,           {magnitude of wind vector}
as,             {UAV's air speed}
numdays,        {number of days to run random scenarios}
day,            {index of current day}

nvInit,         {# vehicles in initial tour read from a file}
nvu,            {# vehicles currently in use}

windques,       {ask whether or not you want random winds}
magseed, dirseed,        {seeds for random winds}
startques,      {ask whether or not you want to input the initial tour}
servseed, loitseed,      {seeds for random service times}
servques,       {ask whether or not you want random service times}

lowdeg,         {low end of range of wind direction to test}
highdeg, {high end of range of wind direction to test}
lowmag,         {low end of range of wind magnitude to test}
highmag,        {high end of range of wind magnitude to test}
minloiter,
maxloiter,      {minimum & maximum loiter time}

dayscore,       {robustness score of day under consideration}
maxdayscore,    {max robustness measure found}
bestscore,      {robustness measure of best route found}
sumScores,      {sum of all dayscores, used to find a mean}
robustChoice    {tags the resulting tour chosen as most robust}
        : INTEGER;

outfile,  {name of output file}
where,          {where in the code?}
str,

startfile,
file, filein,   {filenames}
filebegin,
fileout3,
fileout2,
fileout : STRING;

loadprint,               {print load on vehicles}
stepprint,               {print each move evaluation}
moveprint,               {print every insert move made by RTS}
startprint,              {print starting tour and tw reduction steps}
cycleprint,              {print hash results}
timeprint,               {print time matrix}
twrdprint : BOOLEAN;     {print tw reduction steps}

psurv     : arrRealType;          {prob of survival array}

coord     : coordArrType;                 {coordinates array}

bfTour,                                   {best feasible tour found}
bestTour,                                 {node array holding best tour}
```

```
        tour,                                       {node array holding the tour}
        inittour   : tourType;                      {tour to read in an initial tour}

        tourPen    : vrpPenType;            {record of curr tour penalties}

        windmag,                            {array of wind magnitude per day}
        winddir,                    {array of wind direction per day}
        duration,                   {array of time to best solution per day}
        besttype,                   {array tracking type of best: 1=feas, 0=not}
        scores,                             {array of robustness scores}

        m,                                  {array of TW midpoints}
        slo, shi,                   {arrays of service time ranges}
        s         : arrIntType;     {service times used}

        dist       : arrReal2dimType;               {no wind distance matrix}

        temp,

        routefreq,                          {counts the frequency that route i to j
                                             is chosen, where i and j are the array
                                             indices, in that order}
        time       : arrInt2dimType;                {time matrix}

        tourChoice : ARRAY INTEGER OF tourType; {array of tour choices per day}


BEGIN

        {INITIALIZE}
        startprint := FALSE;        {print starting tour}
        timeprint := FALSE;         {print time matrix}
        stepprint := FALSE;         {print each RTS step eval}
        moveprint := FALSE;         {print each RTS insert move}
        twrdprint := FALSE;         {print TW reduction steps}
        cycleprint := FALSE;        {print cycle/nocycle steps}
        loadprint := FALSE;         {print quantity & vehicle loads}

OUTPUT(" ");
OUTPUT("Please input the problem to work on:");
INPUT(file);

        NEW(instrm);                                        {open problem file}
        NEW(outstrm);                                       {open results file}
        NEW(outInit);                               {open file for future initial tour}
        filein := file + ".DAT";
        fileout := file + ".OUT";
        fileout2 := file + "Init.OUT";
        ASK instrm Open(filein, Input);
        ASK outstrm Open(fileout, Output);
        ASK outInit Open(fileout2, Output);

        fileout3 := file + "Rslt" + ".OUT";
        NEW(outstrm2);
        ASK outstrm2 Open(fileout3, Output);
```

```
str := "FILE: " + file;
ASK outstrm WriteString(str); ASK outstrm WriteLn; ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input the factor (such as 1, 10, 100, etc.) necessary to convert");
OUTPUT("the time window info to integer quantities");
INPUT(factor);

str := "Factor used for target windows and distances " + REALTOSTR(factor);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input the number of vehicles");
INPUT(nv);


        NEW(timeMatrix);


OUTPUT(" ");
OUTPUT("Do you want random service times?");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(servques);

IF servques = 1

  OUTPUT(" ");
  OUTPUT("Input seed number to use for service time randomization");
  INPUT(servseed);
  OUTPUT(" ");
  OUTPUT("Input seed number to use for loiter randomization");
  INPUT(loitseed);

  NEW(randObj3); NEW(randObj4);
  ASK randObj3 SetSeed(FetchSeed(loitseed));
  ASK randObj4 SetSeed(FetchSeed(servseed));

  OUTPUT(" ");
  OUTPUT("Give the probability you will loiter over a target");
  INPUT(ploiter);

ASK outstrm WriteLn;
str := "loitseed="+INTTOSTR(loitseed)+" servseed="+INTTOSTR(servseed)+
      "  Pr{loiter} = "+REALTOSTR(ploiter);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

        {Reads in a Latitude and Longitude scenario with Service time ranges}
        ASK timeMatrix readLatLongLoiter(instrm, nc, numnodes, factor, nv, coord,
                                         tour, slo, shi, outstrm, startprint);

        NEW(s, 0..nc);
        s[0] := 0;
```

```
ELSE

        {Reads in a Latitude and Longitude scenario: the number of targets,
         the probablities of survival, and the target coordinates}
        ASK timeMatrix readLatLong(instrm, nc, numnodes, factor, nv, coord,
                                   tour, s, outstrm, startprint);


END IF;

        ASK instrm Close;          DISPOSE(instrm);

        {Compute 2 dimensional distance matrix given Latitude and Longitude coords}
        {Does not take wind into account}
        {Does not assume the problem is symmetric, but makes it so}
        ASK timeMatrix distLatLong(nc, numnodes, coord, dist, startprint, outstrm);

IF startprint
{output distance matrix}
NEW(temp, 0..numnodes, 0..numnodes);
where := "No wind distance Matrix complete";
FOR i := 0 TO numnodes
 FOR j := i+1 TO numnodes
   temp[i][j] := TRUNC(dist[i][j]);
   temp[j][i] := temp[i][j];
 END FOR;
END FOR;
timeToFile(where, outstrm, temp, numnodes);
DISPOSE(temp);
END IF;

        mindist := 9999.0; maxdist := 0.0;
        sumTij := 0.0; distAvg := 0.0;
        FOR i := 0 TO nc
          FOR j := i+1 TO nc
            sumTij := sumTij + dist[i][j];
            IF (dist[i][j] < mindist) AND (dist[i][j] > 0.0)
                        mindist := dist[i][j]; END IF;
            IF dist[i][j] > maxdist
                        maxdist := dist[i][j]; END IF;
          END FOR;
        END FOR;

        distAvg := sumTij / (FLOAT((nc+1)*(nc+1))/2.0 - FLOAT(nc+1) );

        OUTPUT(" ");
        OUTPUT("Average distance to travel is ", distAvg);
        OUTPUT("Min distance to travel is ", mindist);
        OUTPUT("Max distance to travel is ", maxdist);
        str := "Average distance to travel is " + REALTOSTR(distAvg);
        ASK outstrm WriteString(str); ASK outstrm WriteLn;
        str := "Min distance to travel is " + REALTOSTR(mindist);
        ASK outstrm WriteString(str); ASK outstrm WriteLn;
        str := "Max distance to travel is " + REALTOSTR(maxdist);
        ASK outstrm WriteString(str); ASK outstrm WriteLn;
```

```
OUTPUT(" ");
OUTPUT("Please input vehicle's air speed (in mi/hr)");
INPUT(as);

OUTPUT(" ");
OUTPUT("Please input the conversion factor to use with the WIND time matrix");
OUTPUT("The time windows will be updated to ensure the conversion matches");
OUTPUT("  (must be at least as great as previous factor)");
INPUT(windconv);

        {Update tour with windconv to match times}
        FOR i := 0 TO numnodes
                IF i <= nc
                        slo[i] := TRUNC(windconv / factor * FLOAT(slo[i]));
                        slo[i] := TRUNC(windconv / factor * FLOAT(slo[i]));
                END IF;

                tour[i].ea := TRUNC(windconv / factor * FLOAT(tour[i].ea));
                tour[i].la := TRUNC(windconv / factor * FLOAT(tour[i].la));

                IF tour[i].type = 2
                  tour[i].arr := tour[i].ea;
                  tour[i].dep := tour[i].arr;
                END IF;

        END FOR;

str := "Air speed " + REALTOSTR(as);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
str := "Factor used to make the wind time matrix integer" + REALTOSTR(windconv);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
ASK outstrm WriteLn;


OUTPUT(" ");
OUTPUT("Please input the number of tabu search iterations");
OUTPUT("you would like to step through.");
INPUT(iters);

ASK outstrm WriteLn;
str :="# Iters = " + INTTOSTR(iters);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input the number of days for which you would like to ");
OUTPUT("test random scenarios.");
INPUT(numdays);

OUTPUT(" ");
OUTPUT("Do you want random wind effects on every day");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(windques);
```

```
IF windques = 1

  OUTPUT(" ");
  OUTPUT("Input seed number to use for wind mag");
  INPUT(magseed);
  OUTPUT(" ");
  OUTPUT("Input seed number to use for wind dir");
  INPUT(dirseed);

ASK outstrm WriteLn;
str := "magseed="+INTTOSTR(magseed)+" dirseed="+INTTOSTR(dirseed);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

  NEW(randObj1);  ASK randObj1 SetSeed(FetchSeed(magseed));
  NEW(randObj2);  ASK randObj2 SetSeed(FetchSeed(dirseed));

  OUTPUT(" ");
  OUTPUT("Please input the range of DEGREES you would like to test");
  OUTPUT("  - Put lowest number first");
  OUTPUT("  - If testing winds around the 0 deg direction,");
  OUTPUT("     Make sure lowdeg is negative");
  INPUT(lowdeg);
  INPUT(highdeg);

  OUTPUT(" ");
  OUTPUT("Please input the range of MAGNITUDE you would like to test");
  OUTPUT("  - Put lowest number first");
  INPUT(lowmag);
  INPUT(highmag);

ASK outstrm WriteLn;
str :="RANDOM WINDS: degrees " + INTTOSTR(lowdeg) + " " + INTTOSTR(highdeg)
        + " magnitude " + INTTOSTR(lowmag) + " " + INTTOSTR(highmag);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

ELSE

  OUTPUT(" ");
  OUTPUT("Please input the magnitude of the wind vector (in mi/hr)");
  INPUT(wmag);

  OUTPUT(" ");
  OUTPUT("Please input the direction that the wind is blowing FROM in degrees");
  OUTPUT("  (due EAST is 0 degs, due NORTH is 90 degs, and so on)");
  INPUT(wdir);

  wdir := pi / 180.0 * wdir;

END IF;

OUTPUT(" ");
OUTPUT("Do you want to input the initial tour?");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(startques);
```

```
IF startques = 1

  OUTPUT(" ");
  OUTPUT("Input the file from which to read the initial tour");
  INPUT(startfile);

  {open problem file}
  NEW(instrm2);

  filein := startfile + ".DAT";
  ASK instrm2 Open(filein, Input);

  {initialize array of node id's}
  NEW(m, 0..numnodes);
  FOR j := 1 TO nc
          m[j] := 0;
  END FOR;

  ASK instrm2 ReadInt(nvInit);

  IF nvInit <> nv
    OUTPUT("nv and # vehicles in initial tour do not agree -- Break program!!");
  END IF;

  FOR i := 0 TO numnodes
    ASK instrm2 ReadInt(m[i]);  {m contains the id at position i}
  END FOR;

  ASK instrm2 Close;        DISPOSE(instrm2);

END IF;


OUTPUT(file);
OUTPUT(filein);
OUTPUT(fileout);
OUTPUT(fileout2);

{*} {denotes a parameter setting}
{*        nv := 10;              *}
{*        windconv := 10.0; *}
{*        numcycles := 3;     *}
{*        iters := 1000;    *}
{*}       TWPEN := 1.0;
{*}       gamma := 0;

{*}       INCREASE := 1.2;
{*}       DECREASE := 0.9;
{*}       CYMAX := 50;
{*}       HTSIZE := 1009;
{*}       ZRANGE := 1009;
{*}       minTL := 5;
{*}       maxTL := 2000;
```

```
{*}     DEPTH := nc+nv-1;
{*}     tabuLen := MINOF(30, nc+nv-1);

        {**** LOOP OF SCENARIOS ***}

        NEW(windmag, 1..numdays);
        NEW(winddir, 1..numdays);
        NEW(duration, 1..numdays);
        NEW(besttype, 1..numdays);
        NEW(scores, 1..numdays);
        NEW(tourChoice, 1..numdays, 0..numnodes);
        NEW(routefreq, 0..numnodes, 0..numnodes);

        {initialize matrix of route frequency counts}
        FOR i := 0 TO numnodes
         FOR j := 0 TO numnodes
               routefreq[i][j] := 0;
         END FOR;
        END FOR;

        FOR day := 1 TO numdays

ASK outstrm WriteLn;
str :="DAY: " + INTTOSTR(day);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

          IF windques = 1
           windmag[day] := ASK randObj1 UniformInt(lowmag, highmag);
           wmag := windmag[day];
           winddir[day] := ASK randObj2 UniformInt(lowdeg, highdeg);
           wdir := FLOAT(winddir[day]);
           wdir := pi / 180.0 * wdir;
          END IF;

ASK outstrm WriteLn;
str :="WIND: magnitude ="+INTTOSTR(wmag)+" direction(rads) = " + REALTOSTR(wdir);
ASK outstrm WriteString(str); ASK outstrm WriteLn; ASK outstrm WriteLn;

          IF servques = 1

            FOR i := 1 TO nc

              loiter := ASK randObj3 UniformReal(0.0, 1.0);

              IF loiter <= ploiter
                s[i] := ASK randObj4 UniformInt(slo[i], shi[i]);
              ELSE
                s[i] := slo[i];
              END IF;

            END FOR;

          END IF;
```

```
          ASK timeMatrix timeMatrix(nc, numnodes, gamma, as, wmag, wdir, windconv,
                                      coord, s, dist, time, outstrm, startprint);
OUTPUT("b");

          NEW(startTour);          {find initial tour and/or initial penalties}

          IF day = 1 {must find a true initial tour}

            IF startques = 1         {read in initial tour}

                NEW(inittour, 0..numnodes);

                {Reorder tour, currently in numerical order, by the initial tour
                 and place temporarily into inittour}

                FOR i := 0 TO numnodes
                  inittour[i] := CLONE(tour[m[i]]);
                END FOR;

                {Copy inittour into tour}
                FOR i := 0 TO numnodes
                  tour[i] := CLONE(inittour[i]);
                END FOR;

                DISPOSE(inittour);

                tourSched(1, nc, numnodes, tour, time, tourLen, outstrm);

                startTime := SystemTime();

            ELSE

              ASK startTour startTour(nv, nc, time, tour, tourLen,
                                      totPenalty, tourhv, startTime, m, outstrm);
            END IF;

            DISPOSE(m);
OUTPUT("c");
          ELSE

            NEW(tour, 0.. numnodes);
            {use the best result of the previous day}
        FOR i := 0 TO numnodes
          tour[i] := CLONE(tourChoice[day-1][i]);
        END FOR;

            {Compute initial schedule, return tour's total travel + wait time}
            tourSched(1, nc, numnodes, tour, time, tourLen, outstrm);

            startTime := SystemTime();

       END IF;
OUTPUT("d");
IF startprint
ASK outstrm WriteString("startTour complete"); ASK outstrm WriteLn;
```

```
qcktourFile(outstrm, tour, numnodes);
END IF;
          ASK startTour startPenBest(numnodes, tvl, tourLen, tour, TWPEN,
                                     totPenalty, penTrav, tourCost, tourPen,
                                     bfiter, bfCost, bfTT, bfnv, bestiter,
                                     bestCost, bestTT, bestnv, bestTimeF,
                                     bestTime, bestTour, bfTour);
OUTPUT("e");
          NEW(rts);
          {conduct RTS}
          ASK rts search(TWPEN, INCREASE, DECREASE, HTSIZE, CYMAX, ZRANGE, DEPTH,
                         minTL, maxTL, tabuLen, iters, nc, numnodes,
                         outstrm, outstrm2, tourPen, time, stepprint,
                         moveprint, cycleprint, tourCost, penTrav,
                         totPenalty, tvl, bfCost, bfTT, bfnv, bfiter,
                         bestCost, bestTT, bestnv, bestTime, bestTimeF,
                         bestiter, numfeas, tour, bestTour, bfTour);
          DISPOSE(rts);

          stopTime := SystemTime();

          IF bfiter > -1

           {save the best feasible tour found}
           FOR i := 0 TO numnodes
             tourChoice[day][i] := CLONE(bfTour[i]);
           END FOR;

           {output the results}
           where := "DAY " + INTTOSTR(day) + "  BEST FEASIBLE TOUR";
           twServToFile(where, outstrm, bfTour, nc, numnodes, bfCost,
                    windconv, loadprint, s, slo, shi);

           duration[day] := bestTimeF - startTime;
           besttype[day] := 1;

           ASK outstrm WriteString("# vehicles used = ");
           ASK outstrm WriteInt(bfnv, 2); ASK outstrm WriteLn;

           ASK outstrm WriteString("Best Feasible solution found after ");
           ASK outstrm WriteString(INTTOSTR(bestTimeF-startTime)+" secs");
           ASK outstrm WriteLn;
           ASK outstrm WriteString("on Iteration: "+ INTTOSTR(bfiter));
           ASK outstrm WriteLn;
           ASK outstrm WriteString("with travel time = "+ INTTOSTR(bfTT));
           ASK outstrm WriteLn;

           {update the route frequency matrix}
           FOR i := 0 TO numnodes-1
                 j := bfTour[i].id;
                 k := bfTour[i+1].id;
                 routefreq[j][k] := routefreq[j][k] + 1;
           END FOR;

OUTPUT("f");
```

```
ELSE

  {save the best tour found}
  FOR i := 0 TO numnodes
    tourChoice[day][i] := CLONE(bestTour[i]);
  END FOR;

  {output the results}
  where := "DAY " + INTTOSTR(day)
               + "  Search complete: BEST TOUR (NOT FEASIBLE)";
  twServToFile(where, outstrm, bestTour, nc, numnodes, bestCost,
          windconv, loadprint, s, slo, shi);

  duration[day] := bestTime - startTime;
  besttype[day] := 0;

  ASK outstrm WriteString("# vehicles used = ");
  ASK outstrm WriteInt(bestnv, 2); ASK outstrm WriteLn;
  ASK outstrm WriteString("Best solution found after ");
  ASK outstrm WriteString(INTTOSTR(bestTime-startTime)+" secs");
  ASK outstrm WriteLn;
  ASK outstrm WriteString("on Iteration: "+ INTTOSTR(bestiter));
  ASK outstrm WriteLn;
  ASK outstrm WriteString("with travel time = "+ INTTOSTR(bestTT));
  ASK outstrm WriteLn;
{**** DONT UPDATE THE MATRIX WITH A BAD TOUR
  {update the route frequency matrix}
  FOR i := 0 TO numnodes-1
      j := bestTour[i].id;
      k := bestTour[i+1].id;
      routefreq[j][k] := routefreq[j][k] + 1;
  END FOR;
*****}
END IF;

{output coords to file so we can scatter plot tours}
where := "Day = " + INTTOSTR(day);
LatLongToFile(where, outstrm2, tourChoice[day], nc, numnodes, coord);

{Output service time difference}
IF servques = 1
  servSum := 0.0;
  FOR i := 1 TO nc
    servSum := servSum + FLOAT(s[i] - slo[i]);
  END FOR;
  servSum := servSum / windconv;
  str :="Sum of increase over min service times = "+ REALTOSTR(servSum);
  ASK outstrm WriteLn; ASK outstrm WriteString(str);
  ASK outstrm WriteLn;
END IF;

DISPOSE(bfTour);
DISPOSE(bestTour);
DISPOSE(tourPen);
DISPOSE(tour);
```

```
        END FOR;
        {**** END OF DAY LOOP ****}
OUTPUT("g");

        {output route frequency matrix}
        where := "SCENARIO LOOP COMPLETE, Frequency of Routes Chosen: ";
        timeToFile(where, outstrm, routefreq, numnodes);

        {find most robust tour chosen}
        dayscore := 0;
        maxdayscore := 0;
        sumScores := 0;
        robustChoice := 1;

        FOR day := 1 TO numdays

          FOR i := 0 TO numnodes-1
            dayscore := dayscore
                  + routefreq[tourChoice[day][i].id][tourChoice[day][i+1].id];
          END FOR;
          scores[day] := dayscore;
          sumScores := sumScores + dayscore;

          {choose the tour with the most robust routes}
          IF dayscore > maxdayscore
            robustChoice := day;
            bestscore := dayscore;
            maxdayscore := dayscore;

          ELSIF dayscore = maxdayscore

            {choose feasible tours over nonfeas, or choose the most recent}
            IF besttype[day] >= besttype[robustChoice]
              robustChoice := day;
              bestscore := dayscore;
            END IF;

          END IF;

          dayscore := 0;

        END FOR;
OUTPUT("1");
        {output robust tour}
        tourSched(1, nc, numnodes, tourChoice[robustChoice], time, tourLen, outstrm);
        countVeh(numnodes, tourChoice[robustChoice], nvu);
OUTPUT("2");
        where := "MOST ROBUST TOUR: day = " + INTTOSTR(robustChoice);
        twServToFile(where, outstrm, tourChoice[robustChoice],
                nc, numnodes, tourLen, windconv, loadprint, s, slo, shi);

        ASK outstrm WriteString("# vehicles used = ");
        ASK outstrm WriteInt(nvu, 2); ASK outstrm WriteLn;
        ASK outstrm WriteString("With robustness score "+ INTTOSTR(bestscore));
```

```
        ASK outstrm WriteLn; ASK outstrm WriteLn;
OUTPUT("3");
        ASK outstrm WriteString("MEAN robustness score "
                        + INTTOSTR(sumScores DIV numdays));
        ASK outstrm WriteLn; ASK outstrm WriteLn;

        {Output Robustness scores}
        ASK outstrm WriteLn;
        ASK outstrm WriteString("Robustness scores: ");
        FOR i := 1 TO numdays
          ASK outstrm WriteInt(i, 3);
          ASK outstrm WriteInt(scores[i], 5);
          ASK outstrm WriteLn;
        END FOR;

        {Output Robust tour for future Initial tour}
        ASK outInit WriteInt(nv, 3); ASK outInit WriteLn;
        FOR i:= 0 TO numnodes
          ASK outInit WriteInt(tourChoice[robustChoice][i].id, 5);
          ASK outInit WriteLn;
        END FOR;

        ASK outstrm Close;
        ASK outInit Close;
    ASK outstrm2 Close;

        DISPOSE(startTour);
        DISPOSE(timeMatrix);
        DISPOSE(outstrm);
        DISPOSE(outInit);
        DISPOSE(outstrm2);
        DISPOSE(s);
        DISPOSE(time);
        DISPOSE(coord);

        IF windques = 1
        DISPOSE(randObj1); DISPOSE(randObj2);
        END IF;
        IF servques = 1
        DISPOSE(randObj3); DISPOSE(randObj4);
        END IF;

END MODULE; {MAIN}
```

Appendix H: MuavThreat2

The main module MuavThreat2 runs the *initialization phase* of UAV problems with stochastic winds, service times, and threats. The threats adjust by -0.1, 0.0, or 0.1 and every target is open to having a threat adjustment.

```
MAIN MODULE uavThreat2;

FROM IOMod IMPORT StreamObj, ALL FileUseType, ReadKey;
FROM OSMod IMPORT SystemTime;
FROM MathMod IMPORT pi;

FROM uavMod IMPORT timeMatrixObj;
FROM twReduceMod IMPORT twReductionObj;
FROM uavMod IMPORT startUAVObj;
FROM uavMod IMPORT uavRTSobj;     {risk oriented tabu search}

FROM tabuMod IMPORT coordArrType;
FROM tabuMod IMPORT tourType;
FROM tabuMod IMPORT nodeType;
FROM tabuMod IMPORT vrpPenType;
FROM tabuMod IMPORT arrInt2dimType;
FROM tabuMod IMPORT arrReal2dimType;
FROM tabuMod IMPORT arrIntType;
FROM tabuMod IMPORT arrRealType;

FROM uavMod IMPORT twCvrgServToFile;
FROM tabuMod IMPORT LatLongToFile;
FROM tabuMod IMPORT qcktourFile;
FROM tabuMod IMPORT timeToFile;
FROM tabuMod IMPORT tourSched;
FROM tabuMod IMPORT countVeh;

FROM uavMod IMPORT expCvrg;

FROM RandMod IMPORT RandomObj, SetSeed, FetchSeed;

VAR
        timeMatrix : timeMatrixObj;
        twReduce : twReductionObj;
        startTour : startUAVObj;
        rts : uavRTSobj;
        randObj1, randObj2, randObj3, randObj4, randObj5 : RandomObj;

        instrm,
        instrm2,
        outstrm,
        outstrm2,
        outInit : StreamObj;
```

```
factor,   {used to convert the time windows to integer values}
TWPEN,          {Penalty weight assigned to the sum of late arr TW violations}
INCREASE,       {RTS parameter: mult. factor to decrease tabu length}
DECREASE,       {RTS parameter: mult. factor to increase tabu length}

windconv,   {multiplied by the resulting UAV time matrix, it provides an
             integer matrix (for calc speed) with the needed precision}
sumTij,         {sum of the i to j distances in the distance matrix}
mindist, {minimum travel distance}
maxdist, {maximum travel distance}
distAvg, {avg travel distance}
wdir,           {direction of wind vector}
riskadj,  {amount to randomly adjust a target's prob of survival}
PSFCT,          {factor multiplied by coverage results to get more info into
                 the integer move value}
cvrg,           {expected coverage of the tour}
bfCvrg,  {exp coverages of best and best feas tours}
bestCvrg,
loiter,   {use to increase random service time is exceeded}
ploiter,
servSum {for output of the increase in service times}
   : REAL;

i, j, k,
endnum,         {end number in a numbered data file group}
maxtime,        {max possible time of arrival to any node, for time read}
numcycles,      {number of TW reduction cycles wanted}
numchanges,     {number of TWs reduced by TW reduction Obj}
numnodes,       {number of nodes in the directed graph}
nv,             {number of vehicles}
nc,             {number of targets/customers}
gamma,          {arbitrary cost assigned to the use of each vehicle}
iters,          {number of Tabu Search Iterations per problem}
tourLen, {Length of tour in time}
tvl,            {travel time of tour}
totPenalty,     {Total Penalty assigned to current tour}
tourCost,       {tour Length + Time Window Cost}
penTrav,        {tourCost - totWait == travel time + TW penalty}
bfCost,         {lowest tourCost found for a feasible tour}
bestCost,       {lowest tourCost found for a any tour}
bestTT,         {lowest travel time found for a any tour}
bestnv,         {# vehs used by best overall tour}
bfTT,           {lowest travel time found for feasible tour}
bfnv,           {# vehs used by best feas tour}
bfiter,   {iteration # when best feasible tour found}
tourhv,         {tour's hashing value}
bestiter, {iteration the best Tour found}
bestTime,       {Time the best Tour found}
bfTime, {Time the best feasible Tour found}
numfeas,        {number of feasible solns found}
startTime,      {start Time (after time matrix, before TW reductions)}
stopTime,       {stop Time (after last iteration)}

DEPTH,          {depth of nodes we look for insert moves}
ZRANGE,                 {upper bound on random integer weights assigned to nodes}
```

```
HTSIZE,                 {size of hash table array}
CYMAX,                  {max cyleLength used to alter mavg}
tabuLen, {current length of tabu tenure}
minTL,          {minimum Tabu Length}
maxTL,          {maximum Tabu Length}

wmag,           {magnitude of wind vector}
as,             {UAV's air speed}
numdays,        {number of days to run random scenarios}
day,            {index of current day}

windques,       {ask whether or not you want random winds}
magseed, dirseed,
startques,      {ask whether or not you want to input the initial tour}

lowdeg,         {low end of range of wind direction to test}
highdeg, {high end of range of wind direction to test}
lowmag,         {low end of range of wind magnitude to test}
highmag,        {high end of range of wind magnitude to test}
riskques,{ask whether or not you want random threats}
cvrseed,

nvInit,         {# vehicles in initial tour read from a file}
nvu,            {# vehicles used in current tour}

dayscore,       {robustness score of day under consideration}
maxdayscore,    {max robustness measure found}
bestscore,      {robustness measure of best route found}
sumScores,      {sum of all dayscores, used to find a mean}
robustChoice,   {tags the resulting tour chosen as most robust}

servques,               {ask whether or not you want random service times}
servseed,               {seeds for random service times}
loitseed
        : INTEGER;

outfile,  {name of output file}
where,          {where in the code?}
str,

startfile,
file, filein,
filebegin,
fileout3,
fileout2,
fileout : STRING;       {filenames}

loadprint,              {print load on vehicles}
stepprint,              {print each move evaluation}
moveprint,              {print every insert move made by RTS}
startprint,             {print starting tour and tw reduction steps}
cycleprint,             {print hash results}
timeprint,              {print time matrix}
twrdprint : BOOLEAN;    {print tw reduction steps}
```

```
        psurv      : arrRealType;              {prob of survival array}

        coord      : coordArrType;                        {coordinates array}

        bfTour,                                           {best feasible tour found}
        bestTour,                                         {node array holding best tour}
        tour,                                             {node array holding the tour}
        inittour   : tourType;                            {tour to read in an initial tour}

        tourPen    : vrpPenType;               {record of curr tour penalties}

        windmag,                               {array of wind magnitude per day}
        winddir,                    {array of wind direction per day}
        duration,                   {array of time to best solution per day}
        besttype,                   {array tracking type of best: 1=feas, 0=not}
        scores,                                {array of robustness scores}

        m,                                                {array of TW midpoints}
        slo, shi,                              {low, high ranges for random service}
        s          : arrIntType;               {array of service times}

        dist       : arrReal2dimType;                     {no wind distance matrix}

        temp,

        routefreq,                             {counts the frequency that route i to j
                                                is chosen, where i and j are the array
                                                indices, in that order}
        time       : arrInt2dimType;                      {time matrix}

        tourChoice : ARRAY INTEGER OF tourType; {array of tour choices per day}
        psurvDay : ARRAY INTEGER OF arrRealType; {array of psurv per day}


BEGIN

        {INITIALIZE}
        startprint := FALSE;        {print starting tour}
        timeprint := FALSE;         {print time matrix}
        stepprint := FALSE;         {print each RTS step eval}
        moveprint := FALSE;         {print each RTS insert move}
        twrdprint := FALSE;         {print TW reduction steps}
        cycleprint := FALSE;        {print cycle/nocycle steps}
        loadprint := FALSE;         {print quantity & vehicle loads}

        NEW(outstrm);

OUTPUT(" ");
OUTPUT("Please input the problem to work on:");
INPUT(file);

        NEW(instrm);                                        {open problem file}
        NEW(outstrm);                                       {open results file}
        NEW(outInit);                             {open file for future initial tour}
        filein := file + ".DAT";
```

```
            fileout := file + ".OUT";
            fileout2 := file + "Init.OUT";
            ASK instrm Open(filein, Input);
            ASK outstrm Open(fileout, Output);
            ASK outInit Open(fileout2, Output);

            fileout3 := file + "Rslt" + ".OUT";
            NEW(outstrm2);
            ASK outstrm2 Open(fileout3, Output);

    str := "FILE: " + file;
    ASK outstrm WriteString(str); ASK outstrm WriteLn; ASK outstrm WriteLn;

    OUTPUT(" ");
    OUTPUT("Please input the factor (such as 1, 10, 100, etc.) necessary to convert");
    OUTPUT("the time window info to integer quantities");
    INPUT(factor);

    str := "Factor used for target windows and distances " + REALTOSTR(factor);
    ASK outstrm WriteString(str); ASK outstrm WriteLn;

    OUTPUT(" ");
    OUTPUT("Please input the number of vehicles");
    INPUT(nv);

            NEW(timeMatrix);

    OUTPUT(" ");
    OUTPUT("Do you want random service times?");
    OUTPUT("  - 1 = YES");
    OUTPUT("  - 0 = NO");
    INPUT(servques);

    IF servques = 1

      OUTPUT(" ");
      OUTPUT("Input seed number to use for service time randomization");
      INPUT(servseed);
      OUTPUT(" ");
      OUTPUT("Input seed number to use for loiter randomization");
      INPUT(loitseed);

      NEW(randObj3); NEW(randObj4);

      ASK randObj3 SetSeed(FetchSeed(loitseed));
      ASK randObj4 SetSeed(FetchSeed(servseed));

      OUTPUT(" ");
      OUTPUT("Give the probability you will loiter over a target");
      INPUT(ploiter);

    ASK outstrm WriteLn;
    str := "loitseed="+INTTOSTR(loitseed)+" servseed="+INTTOSTR(servseed)+
          "  Pr{loiter} = "+REALTOSTR(ploiter);
    ASK outstrm WriteString(str); ASK outstrm WriteLn;
```

```
        {Reads in a coords in miles scenario with Service time ranges and psurv}
        ASK timeMatrix readUAVloiter(instrm, nc, numnodes, factor, nv,
                                    psurv, coord, tour, slo, shi,
                                    outstrm, startprint);


        NEW(s, 0..nc);
        s[0] := 0;

ELSE

        {reads UAV file, finds nc, inits coord & tour}
        ASK timeMatrix readUAV(instrm, nc, numnodes, factor, nv,
                               psurv, coord, tour, s, outstrm,
                                    startprint);



END IF;

        ASK instrm Close;        DISPOSE(instrm);

        {compute distance matrix, given coordinates in miles}
        ASK timeMatrix distMatrix(nc, numnodes, coord, dist, outstrm);

IF startprint
{output distance matrix}
NEW(temp, 0..numnodes, 0..numnodes);
where := "No wind distance Matrix complete";
FOR i := 0 TO numnodes
  FOR j := i+1 TO numnodes
    temp[i][j] := TRUNC(dist[i][j]);
    temp[j][i] := temp[i][j];
  END FOR;
END FOR;
timeToFile(where, outstrm, temp, numnodes);
DISPOSE(temp);
END IF;
        mindist := 9999.0; maxdist := 0.0;
        sumTij := 0.0; distAvg := 0.0;
        FOR i := 0 TO nc
          FOR j := i+1 TO nc
            sumTij := sumTij + dist[i][j];
            IF (dist[i][j] < mindist) AND (dist[i][j] > 0.0)
                        mindist := dist[i][j]; END IF;
            IF dist[i][j] > maxdist
                        maxdist := dist[i][j]; END IF;
          END FOR;
        END FOR;

        distAvg := sumTij / (FLOAT((nc+1)*(nc+1))/2.0 - FLOAT(nc+1) );

        OUTPUT(" ");
        OUTPUT("Average distance to travel is ", distAvg);
        OUTPUT("Min distance to travel is ", mindist);
```

```
        OUTPUT("Max distance to travel is ", maxdist);
        str := "Average distance to travel is " + REALTOSTR(distAvg);
        ASK outstrm WriteString(str); ASK outstrm WriteLn;
        str := "Min distance to travel is " + REALTOSTR(mindist);
        ASK outstrm WriteString(str); ASK outstrm WriteLn;
        str := "Max distance to travel is " + REALTOSTR(maxdist);
        ASK outstrm WriteString(str); ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input vehicle's air speed (in mi/hr)");
INPUT(as);

OUTPUT(" ");
OUTPUT("Please input the conversion factor to use with the WIND time matrix");
OUTPUT("The time windows will be updated to ensure the conversion matches");
OUTPUT("  (must be at least as great as previous factor)");
INPUT(windconv);

        {Update tour with windconv to match times}
        FOR i := 0 TO numnodes
                IF i <= nc
                        slo[i] := TRUNC(windconv / factor * FLOAT(slo[i]));
                        shi[i] := TRUNC(windconv / factor * FLOAT(shi[i]));
                END IF;
                tour[i].ea := TRUNC(windconv / factor * FLOAT(tour[i].ea));
                tour[i].la := TRUNC(windconv / factor * FLOAT(tour[i].la));

                IF tour[i].type = 2
                  tour[i].arr := tour[i].ea;
                  tour[i].dep := tour[i].arr;
                END IF;

        END FOR;

str := "Air speed " + REALTOSTR(as);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
str := "Factor used to make the wind time matrix integer" + REALTOSTR(windconv);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
ASK outstrm WriteLn;


OUTPUT(" ");
OUTPUT("Please input the number of tabu search iterations");
OUTPUT("you would like to step through.");
INPUT(iters);

ASK outstrm WriteLn;
str :="# Iters = " + INTTOSTR(iters);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input the number of days for which you would like to ");
OUTPUT("test random scenarios.");
INPUT(numdays);
```

```
OUTPUT(" ");
OUTPUT("Do you want random WIND effects on every day");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(windques);

IF windques = 1

  OUTPUT(" ");
  OUTPUT("Input seed number to use for wind mag");
  INPUT(magseed);
  OUTPUT(" ");
  OUTPUT("Input seed number to use for wind dir");
  INPUT(dirseed);

ASK outstrm WriteLn;
str := "magseed="+INTTOSTR(magseed)+" dirseed="+INTTOSTR(dirseed);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

  NEW(randObj1);  ASK randObj1 SetSeed(FetchSeed(magseed));
  NEW(randObj2);  ASK randObj2 SetSeed(FetchSeed(dirseed));

  OUTPUT(" ");
  OUTPUT("Please input the range of DEGREES you would like to test");
  OUTPUT("  - Put lowest number first");
  OUTPUT("  - If testing winds around the 0 deg direction,");
  OUTPUT("    Make sure lowdeg is negative");
  INPUT(lowdeg);
  INPUT(highdeg);

  OUTPUT(" ");
  OUTPUT("Please input the range of MAGNITUDE you would like to test");
  OUTPUT("  - Put lowest number first");
  INPUT(lowmag);
  INPUT(highmag);

ASK outstrm WriteLn;
str :="RANDOM WINDS: degrees " + INTTOSTR(lowdeg) + " " + INTTOSTR(highdeg)
          + " magnitude " + INTTOSTR(lowmag) + " " + INTTOSTR(highmag);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

ELSE

  OUTPUT(" ");
  OUTPUT("Please input the magnitude of the wind vector (in mi/hr)");
  INPUT(wmag);

  OUTPUT(" ");
  OUTPUT("Please input the direction that the wind is blowing FROM in degrees");
  OUTPUT("  (due EAST is 0 degs, due NORTH is 90 degs, and so on)");
  INPUT(wdir);

  wdir := pi / 180.0 * wdir;

END IF;
```

```
OUTPUT(" ");
OUTPUT("Do you want to input the initial tour?");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(startques);

IF startques = 1

 OUTPUT(" ");
 OUTPUT("Input the file from which to read the initial tour");
 INPUT(startfile);

 NEW(instrm2);                                   {open problem file}

 filein := startfile + ".DAT";
 ASK instrm2 Open(filein, Input);

 {initialize array of node id's}
 NEW(m, 0..numnodes);
 FOR j := 1 TO nc
         m[j] := 0;
 END FOR;

 ASK instrm2 ReadInt(nvInit);

 IF nvInit <> nv
   OUTPUT("nv and # vehicles in initial tour do not agree -- Break program!!");
 END IF;

 FOR i := 0 TO numnodes
   ASK instrm2 ReadInt(m[i]);  {m contains the id at position i}
 END FOR;

 ASK instrm2 Close;       DISPOSE(instrm2);

END IF;


OUTPUT(" ");
OUTPUT("Do you want random THREAT effects on every day");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(riskques);

OUTPUT(" ");
OUTPUT("Input the factor to convert coverage to an integer value");
INPUT(PSFCT);

IF riskques = 1

 OUTPUT(" ");
 OUTPUT("Input seed number to use for random COVERAGES");
 INPUT(cvrseed);
```

```
ASK outstrm WriteLn;
str := "RANDOM THREATS: cvrseed="+INTTOSTR(cvrseed);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

 NEW(randObj5);  ASK randObj5 SetSeed(FetchSeed(cvrseed));

END IF;

OUTPUT(file);
OUTPUT(filein);
OUTPUT(fileout);
OUTPUT(fileout2);

{*} {denotes a parameter setting}
{*      nv := 10;             *}
{*      windconv := 10.0; *}
{*      numcycles := 3;   *}
{*      iters := 1000;  *}
{*}     TWPEN := 10.0;
{*}     gamma := 0;

{*}     INCREASE := 1.2;
{*}     DECREASE := 0.9;
{*}     CYMAX := 50;
{*}     HTSIZE := 131073;
{*}     ZRANGE := 1009;
{*}     minTL := 5;
{*}     maxTL := 2000;


{*}     DEPTH := nc+nv-1;
{*}     tabuLen := MINOF(30, nc+nv-1);

        {**** LOOP OF SCENARIOS ***}

        NEW(windmag, 1..numdays);
        NEW(winddir, 1..numdays);
        NEW(duration, 1..numdays);
        NEW(scores, 1..numdays);
        NEW(besttype, 1..numdays);
        NEW(psurvDay, 1..numdays, 0..nc);
        NEW(tourChoice, 1..numdays, 0..numnodes);
        NEW(routefreq, 0..numnodes, 0..numnodes);

        FOR i := 0 TO numnodes
         FOR j := 0 TO numnodes
              routefreq[i][j] := 0;
         END FOR;
        END FOR;

        FOR day := 1 TO numdays

ASK outstrm WriteLn;
str :="DAY: " + INTTOSTR(day);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
```

```
            IF windques = 1
              windmag[day] := ASK randObj1 UniformInt(lowmag, highmag);
              wmag := windmag[day];
              winddir[day] := ASK randObj2 UniformInt(lowdeg, highdeg);
              wdir := FLOAT(winddir[day]);
              wdir := pi / 180.0 * wdir;
            END IF;

    ASK outstrm WriteLn;
    str :="WIND: magnitude ="+INTTOSTR(wmag)+" direction(rads) = " + REALTOSTR(wdir);
    ASK outstrm WriteString(str); ASK outstrm WriteLn; ASK outstrm WriteLn;

            IF riskques = 1

              {randomly adjust the prob of survival at the target nodes}
              FOR i := 1 TO nc
                riskadj := ASK randObj3 UniformReal(-1.0, 1.0);
                    IF riskadj < -0.333
                            riskadj := -1.0;
                    ELSIF riskadj > 0.333
                            riskadj := 1.0;
           ELSE
                            riskadj := 0.0;
                    END IF;
                psurvDay[day][i] := psurv[i] + riskadj / 10.0;

              END FOR;

            ELSE
                    psurvDay[day] := psurv;
            END IF;

            IF servques = 1

              FOR i := 1 TO nc

                loiter := ASK randObj3 UniformReal(0.0, 1.0);

                IF loiter <= ploiter
                  s[i] := ASK randObj4 UniformInt(slo[i], shi[i]);
                ELSE
                  s[i] := slo[i];
                END IF;

              END FOR;

            END IF;

            ASK timeMatrix timeMatrix(nc, numnodes, gamma, as, wmag, wdir, windconv,
                                        coord, s, dist, time, outstrm, startprint);


            NEW(startTour);          {find initial tour and/or initial penalties}
```

```
IF day = 1 {must find a true initial tour}

  IF startques = 1          {read in initial tour}

      NEW(inittour, 0..numnodes);

      {Reorder tour, currently in numerical order, by the initial tour
       and place temporarily into inittour}

      FOR i := 0 TO numnodes
        inittour[i] := CLONE(tour[m[i]]);
      END FOR;

      {Copy inittour into tour}
      FOR i := 0 TO numnodes
        tour[i] := CLONE(inittour[i]);
      END FOR;

      DISPOSE(inittour);

      tourSched(1, nc, numnodes, tour, time, tourLen, outstrm);

      startTime := SystemTime();

  ELSE

    ASK startTour startTour(nv, nc, time, tour, tourLen,
                            totPenalty, tourhv, startTime, m, outstrm);
  END IF;

  DISPOSE(m);

ELSE

  NEW(tour, 0..numnodes);
  {use the best result of the previous day}
FOR i := 0 TO numnodes
  tour[i] := CLONE(tourChoice[day-1][i]);
END FOR;

  {Compute initial schedule, return tour's total travel + wait time}
  tourSched(1, nc, numnodes, tour, time, tourLen, outstrm);

  startTime := SystemTime();

END IF;

IF startprint
where := "startTour complete";
qcktourFile(outstrm, tour, numnodes);
END IF;

    ASK startTour startUAVbest(numnodes, tvl, tourLen, tour, TWPEN,
                        psurvDay[day], totPenalty, penTrav, tourCost,
                        tourPen, bfiter, bfCost, bfTT, bfnv, bestiter,
```

```
                              bestCost, bestTT, bestnv, bfTime,
                              bestTime, cvrg, bfCvrg, bestCvrg,
                              bestTour, bfTour);

NEW(rts);
{conduct RTS}

ASK rts search(psurvDay[day], PSFCT,
          TWPEN, INCREASE, DECREASE, HTSIZE, CYMAX, ZRANGE, DEPTH,
          minTL, maxTL, tabuLen, iters, nc, numnodes,
          outstrm, outstrm2, tourPen, time, stepprint,
          moveprint, cycleprint, tourCost, penTrav, totPenalty, tvl,
          bfCost, bfTT, bfnv, bfiter, bestCost, bestTT, bestnv,
          bestTime, bfTime, bestiter, numfeas,
          bfCvrg, bestCvrg, cvrg,
          tour, bestTour, bfTour);
DISPOSE(rts);

stopTime := SystemTime();

IF bfiter > -1

 {save the best feasible tour found}
 FOR i := 0 TO numnodes
   tourChoice[day][i] := CLONE(bfTour[i]);
 END FOR;

 {output the results}
 where := "DAY " + INTTOSTR(day) + "  BEST FEASIBLE TOUR";
 twCvrgServToFile(where, outstrm, bfTour, nc, numnodes, bfCost,
                        factor, loadprint, psurvDay[day], s, slo);
 duration[day] := bfTime - startTime;
 besttype[day] := 1;

 ASK outstrm WriteString("# vehicles used = ");
 ASK outstrm WriteInt(bfnv, 2); ASK outstrm WriteLn;

 ASK outstrm WriteString("Best Feasible solution found after ");
 ASK outstrm WriteString(INTTOSTR(bfTime-startTime)+" secs");
 ASK outstrm WriteLn;
 ASK outstrm WriteString("on Iteration: "+ INTTOSTR(bfiter));
 ASK outstrm WriteLn;
 ASK outstrm WriteString("with travel time = "+ INTTOSTR(bfTT));
 ASK outstrm WriteLn;
 ASK outstrm WriteString("& Expected coverage = "+REALTOSTR(bfCvrg));
 ASK outstrm WriteLn;

 {update the route frequency matrix}
 FOR i := 0 TO numnodes-1
      j := bfTour[i].id;
      k := bfTour[i+1].id;
      routefreq[j][k] := routefreq[j][k] + 1;
 END FOR;

ELSE
```

```
            {save the best tour found}
            FOR i := 0 TO numnodes
              tourChoice[day][i] := CLONE(bestTour[i]);
            END FOR;

            {output the results}
            where := "DAY "+ INTTOSTR(day) +"  Search complete: BEST TOUR (NOT FEAS)";
            twCvrgServToFile(where, outstrm, bestTour, nc, numnodes, bestCost,
                                      factor, loadprint, psurvDay[day], s, slo);
            duration[day] := bestTime - startTime;
            besttype[day] := 0;

            ASK outstrm WriteString("# vehicles used = ");
            ASK outstrm WriteInt(bestnv, 2); ASK outstrm WriteLn;

            ASK outstrm WriteString("Best solution found after ");
            ASK outstrm WriteString(INTTOSTR(bestTime-startTime)+" secs");
            ASK outstrm WriteLn;
            ASK outstrm WriteString("on Iteration: "+ INTTOSTR(bestiter));
            ASK outstrm WriteLn;
            ASK outstrm WriteString("with travel time = "+ INTTOSTR(bestTT));
            ASK outstrm WriteLn;
            ASK outstrm WriteString("& Expected coverage = "+REALTOSTR(bestCvrg));
            ASK outstrm WriteLn;
{*** DONT UPDATE THE ROUTE FREQ MATRIX WITH BAD TOURS
            {update the route frequency matrix}
            FOR i := 0 TO numnodes-1
                  j := bestTour[i].id;
                  k := bestTour[i+1].id;
                  routefreq[j][k] := routefreq[j][k] + 1;
            END FOR;
****}
          END IF;

          {output coords to file so we can scatter plot tours}
          where := "Day = " + INTTOSTR(day);
          LatLongToFile(where, outstrm2, tourChoice[day], nc, numnodes, coord);

          {Output service time difference}
          IF servques = 1
            servSum := 0.0;
            FOR i := 1 TO nc
              servSum := servSum + FLOAT(s[i] - slo[i]);
            END FOR;
            servSum := servSum / windconv;
            str :="Sum of increase over min service times = "+ REALTOSTR(servSum);
            ASK outstrm WriteLn; ASK outstrm WriteString(str);
            ASK outstrm WriteLn;
          END IF;

          DISPOSE(bfTour);
          DISPOSE(bestTour);
          DISPOSE(tourPen);
          DISPOSE(tour);
```

```
END FOR; {day loop}
{**** END OF DAY LOOP ****}

{output route frequency matrix}
where := "SCENARIO LOOP COMPLETE, Frequency of Routes Chosen: ";
timeToFile(where, outstrm, routefreq, numnodes);

{find most robust tour chosen}
dayscore := 0;
maxdayscore := 0;
sumScores := 0;
robustChoice := 1;

FOR day := 1 TO numdays

  FOR i := 0 TO numnodes-1
    dayscore := dayscore
           + routefreq[tourChoice[day][i].id][tourChoice[day][i+1].id];
  END FOR;
  scores[day] := dayscore;
  sumScores := sumScores + dayscore;

  {choose the tour with the most robust routes}
  IF dayscore > maxdayscore
    robustChoice := day;
    bestscore := dayscore;
    maxdayscore := dayscore;

  ELSIF dayscore = maxdayscore

    {choose feasible tours over nonfeas, or choose the most recent}
    IF besttype[day] >= besttype[robustChoice]
      robustChoice := day;
      bestscore := dayscore;
    END IF;

  END IF;

  dayscore := 0;

END FOR;

{output robust tour}
tourSched(1, nc, numnodes, tourChoice[robustChoice], time, tourLen, outstrm);
countVeh(numnodes, tourChoice[robustChoice], nvu);
expCvrg(numnodes, psurvDay[robustChoice], tourChoice[robustChoice], cvrg);

where := "MOST ROBUST TOUR: day = " + INTTOSTR(robustChoice);
twCvrgServToFile(where, outstrm, tourChoice[robustChoice], nc, numnodes,
         tourLen, factor, loadprint, psurvDay[day], s, slo);

ASK outstrm WriteString("# vehicles used = ");
ASK outstrm WriteInt(nvu, 2); ASK outstrm WriteLn;
ASK outstrm WriteString("Expected Coverage = ");
```

```
    ASK outstrm WriteReal(cvrg, 6, 1); ASK outstrm WriteLn;
    ASK outstrm WriteString("With robustness score "+ INTTOSTR(bestscore));
    ASK outstrm WriteLn; ASK outstrm WriteLn;

    ASK outstrm WriteString("MEAN robustness score "
                + INTTOSTR(sumScores DIV numdays));
    ASK outstrm WriteLn; ASK outstrm WriteLn;

    {Output Robustness scores}
    ASK outstrm WriteLn;
    ASK outstrm WriteString("Robustness scores: ");
    ASK outstrm WriteLn;
    FOR i := 1 TO numdays
      ASK outstrm WriteInt(i, 3);
      ASK outstrm WriteInt(scores[i], 5);
      ASK outstrm WriteLn;
    END FOR;

    {Output Robust tour for future Initial tour}
    ASK outInit WriteInt(nv, 5); ASK outInit WriteLn;
    FOR i:= 0 TO numnodes
      ASK outInit WriteInt(tourChoice[robustChoice][i].id, 5);
      ASK outInit WriteLn;
    END FOR;

    ASK outstrm Close;
    ASK outInit Close;
  ASK outstrm2 Close;

    DISPOSE(timeMatrix);
    DISPOSE(outstrm);
    DISPOSE(outstrm2);
    DISPOSE(s);
    DISPOSE(time);
    DISPOSE(coord);

    IF windques = 1
    DISPOSE(randObj1); DISPOSE(randObj2);
    END IF;
    IF riskques = 1
    DISPOSE(randObj3);
    END IF;

END MODULE. {MAIN}
```

Appendix I: MuavServ2

A second step to MuavLoiter, the main module MuavServ2 runs the *evaluation phase* of UAV problems with stochastic winds and service times. An entire set of tours and the route frequency matrix from the *initialization phase* is read into the module. Every day in this phase the route frequency matrix is updated and the robust tour is re-identified.

```
MAIN MODULE uavServ2;

FROM IOMod IMPORT StreamObj, ALL FileUseType, ReadKey;
FROM OSMod IMPORT SystemTime;
FROM MathMod IMPORT pi;

FROM uavMod IMPORT timeMatrixObj;
FROM twReduceMod IMPORT twReductionObj;
FROM tsptwMod IMPORT startTourObj;
FROM tsptwMod IMPORT reacTabuObj;

FROM tabuMod IMPORT arrReal2dimType;
FROM tabuMod IMPORT coordArrType;
FROM tabuMod IMPORT tourType;
FROM tabuMod IMPORT nodeType;
FROM tabuMod IMPORT vrpPenType;
FROM tabuMod IMPORT arrInt2dimType;
FROM tabuMod IMPORT arrIntType;
FROM tabuMod IMPORT arrRealType;

FROM tabuMod IMPORT SwapNode;
FROM uavMod IMPORT twServToFile;
FROM tabuMod IMPORT LatLongToFile;
FROM tabuMod IMPORT qcktourFile;
FROM tabuMod IMPORT tourToScreen;
FROM tabuMod IMPORT timeToFile;
FROM tabuMod IMPORT tourSched;
FROM tabuMod IMPORT countVeh;

FROM RandMod IMPORT RandomObj, SetSeed, FetchSeed;

VAR
        timeMatrix : timeMatrixObj;
        twReduce : twReductionObj;
        startTour : startTourObj;
        rts : reacTabuObj;
        randObj1, randObj2, randObj3, randObj4 : RandomObj;

        instrm,
```

```
instrm2, instrm3, instrm4,
outstrm,
outstrm2,
outInit : StreamObj;

factor,    {used to convert the time windows to integer values}
TWPEN,          {Penalty weight assigned to the sum of late arr TW violations}
INCREASE,       {RTS parameter: mult. factor to decrease tabu length}
DECREASE,       {RTS parameter: mult. factor to increase tabu length}

windconv,   {multiplied by the resulting UAV time matrix, it provides an
            integer matrix (for calc speed) with the needed precision}
sumTij,         {sum of the i to j distances in the distance matrix}
mindist, {minimum travel distance}
maxdist, {maximum travel distance}
distAvg, {avg travel distance}
wdir,           {direction of wind vector}
ploiter,  {probabilitiy you loiter over a target}
loiter,         {loiter? - individual node result}

servSum         {sum of increase over minimum service times}
        : REAL;

i, j, k,
endnum,         {end number in a numbered data file group}
maxtime,        {max possible time of arrival to any node, for time read}
numcycles,      {number of TW reduction cycles wanted}
numchanges,     {number of TWs reduced by TW reduction Obj}
numnodes,       {number of nodes in the directed graph}
nv,             {number of vehicles}
nc,             {number of targets/customers}
gamma,          {arbitrary cost assigned to the use of each vehicle}
iters,          {number of Tabu Search Iterations per problem}
tourLen, {Length of tour in time}
tvl,            {travel time of tour}
totPenalty,     {Total Penalty assigned to current tour}
tourCost,       {tour Length + Time Window Cost}
penTrav,        {tourCost - totWait == travel time + TW penalty}
bfCost,         {lowest tourCost found for a feasible tour}
bestCost,       {lowest tourCost found for a any tour}
bestTT,         {lowest travel time found for a any tour}
bestnv,         {# vehs used by best overall tour}
bfTT,           {lowest travel time found for feasible tour}
bfnv,           {# vehs used by best feas tour}
bfiter,   {iteration # when best feasible tour found}
tourhv,         {tour's hashing value}
bestiter, {iteration the best Tour found}
bestTime,       {Time the best Tour found}
bestTimeF,      {Time the best feasible Tour found}
numfeas,        {number of feasible solns found}
startTime,      {start Time (after time matrix, before TW reductions)}
stopTime,       {stop Time (after last iteration)}

DEPTH,          {depth of nodes we look for insert moves}
ZRANGE,                 {upper bound on random integer weights assigned to nodes}
```

```
HTSIZE,                   {size of hash table array}
CYMAX,                    {max cyleLength used to alter mavg}
tabuLen, {current length of tabu tenure}
minTL,          {minimum Tabu Length}
maxTL,          {maximum Tabu Length}

wmag,           {magnitude of wind vector}
as,             {UAV's air speed}
numdays,        {number of days to run random scenarios}
day,            {index of current day}
nInitdays,      {number of days in the initialization set}
totaldays,      {nInitdays + numdays}
nvInit,         {# vehicles in initial tour read from a file}
nvu,            {# vehicles used in current tour}

windques,       {ask whether or not you want random winds}
magseed, dirseed,         {seeds for random winds}
startques,      {ask whether or not you want to input the initial tour}
servseed, loitseed,       {seeds for random service times}
servques,       {ask whether or not you want random service times}
initques, {ask if an initialization set already performed}

lowdeg,         {low end of range of wind direction to test}
highdeg, {high end of range of wind direction to test}
lowmag,         {low end of range of wind magnitude to test}
highmag,        {high end of range of wind magnitude to test}
minloiter,
maxloiter,      {minimum & maximum loiter time}

dayscore,       {robustness score of day under consideration}
maxdayscore,    {max robustness measure found}
bestscore,      {robustness measure of best route found}
sumScores,      {sum of all dayscores, used to find a mean}
robustChoice,   {tags the resulting tour chosen as most robust}

startday, {# of day beginning the scenario, day, loop}
rday     {loop var of the incremental robust tour choice}
         : INTEGER;

outfile,  {name of output file}
where,          {where in the code?}
str,

startfile,
file, filein,   {filenames}
filebegin,
fileout3,
fileout2,
filetour, filefreq,
fileout : STRING;

loadprint,                {print load on vehicles}
stepprint,                {print each move evaluation}
moveprint,                {print every insert move made by RTS}
startprint,               {print starting tour and tw reduction steps}
```

```
        cycleprint,              {print hash results}
        timeprint,               {print time matrix}
        twrdprint : BOOLEAN;     {print tw reduction steps}

        psurv      : arrRealType;          {prob of survival array}

        coord      : coordArrType;                  {coordinates array}

        bfTour,                                     {best feasible tour found}
        bestTour,                                   {node array holding best tour}
        tour,                                       {node array holding the tour}
        oldtour,                           {temporary tour}
        inittour   : tourType;                      {tour to read in an initial tour}

        tourPen    : vrpPenType;           {record of curr tour penalties}

        windmag,                           {array of wind magnitude per day}
        winddir,                 {array of wind direction per day}
        duration,                {array of time to best solution per day}
        besttype,                {array tracking type of best: 1=feas, 0=not}
        scores,                            {array of robustness scores}

        m,                                 {array of TW midpoints}
        slo, shi,                {arrays of service time ranges}
        s          : arrIntType; {service times used}

        dist       : arrReal2dimType;               {no wind distance matrix}

        temp,

        routefreq,                         {counts the frequency that route i to j
                                            is chosen, where i and j are the array
                                            indices, in that order}
        time       : arrInt2dimType;                {time matrix}

        tourChoice : ARRAY INTEGER OF tourType; {array of tour choices per day}

        node : nodeType;

BEGIN

        {INITIALIZE}
        startprint := FALSE;     {print starting tour}
        timeprint := FALSE;      {print time matrix}
        stepprint := FALSE;      {print each RTS step eval}
        moveprint := FALSE;      {print each RTS insert move}
        twrdprint := FALSE;      {print TW reduction steps}
        cycleprint := FALSE;     {print cycle/nocycle steps}
        loadprint := FALSE;      {print quantity & vehicle loads}

OUTPUT(" ");
OUTPUT("Please input the problem to work on:");
INPUT(file);
```

```
        NEW(instrm);                              {open problem file}
        NEW(outstrm);                             {open results file}
        NEW(outInit);                     {open file for future initial tour}
        filein := file + ".DAT";
        fileout := file + ".OUT";
        fileout2 := file + "Init.OUT";
        ASK instrm Open(filein, Input);
        ASK outstrm Open(fileout, Output);
        ASK outInit Open(fileout2, Output);

        fileout3 := file + "Rslt" + ".OUT";
        NEW(outstrm2);
        ASK outstrm2 Open(fileout3, Output);

str := "FILE: " + file;
ASK outstrm WriteString(str); ASK outstrm WriteLn; ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input the factor (such as 1, 10, 100, etc.) necessary to convert");
OUTPUT("the time window info to integer quantities");
INPUT(factor);

str := "Factor used for target windows and distances " + REALTOSTR(factor);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input the number of vehicles");
INPUT(nv);

        NEW(timeMatrix);


OUTPUT(" ");
OUTPUT("Do you want random service times?");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(servques);

IF servques = 1

  OUTPUT(" ");
  OUTPUT("Input seed number to use for service time randomization");
  INPUT(servseed);
  OUTPUT(" ");
  OUTPUT("Input seed number to use for loiter randomization");
  INPUT(loitseed);

  NEW(randObj3); NEW(randObj4);
  ASK randObj3 SetSeed(FetchSeed(loitseed));
  ASK randObj4 SetSeed(FetchSeed(servseed));

  OUTPUT(" ");
  OUTPUT("Give the probability you will loiter over a target");
  INPUT(ploiter);
```

```
ASK outstrm WriteLn;
str := "loitseed="+INTTOSTR(loitseed)+" servseed="+INTTOSTR(servseed)+
     "  Pr{loiter} = "+REALTOSTR(ploiter);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

        {Reads in a Latitude and Longitude scenario with Service time ranges}
        ASK timeMatrix readLatLongLoiter(instrm, nc, numnodes, factor, nv, coord,
                                  tour, slo, shi, outstrm, startprint);

        NEW(s, 0..nc);
        s[0] := 0;

ELSE

        {Reads in a Latitude and Longitude scenario: the number of targets,
         the probablities of survival, and the target coordinates}
        ASK timeMatrix readLatLong(instrm, nc, numnodes, factor, nv, coord,
                                tour, s, outstrm, startprint);


END IF;

        ASK instrm Close;          DISPOSE(instrm);

        {Compute 2 dimensional distance matrix given Latitude and Longitude coords}
        {Does not take wind into account}
        {Does not assume the problem is symmetric, but makes it so}
        ASK timeMatrix distLatLong(nc, numnodes, coord, dist, startprint, outstrm);

IF startprint
{output distance matrix}
NEW(temp, 0..numnodes, 0..numnodes);
where := "No wind distance Matrix complete";
FOR i := 0 TO numnodes
 FOR j := i+1 TO numnodes
   temp[i][j] := TRUNC(dist[i][j]);
   temp[j][i] := temp[i][j];
 END FOR;
END FOR;
timeToFile(where, outstrm, temp, numnodes);
DISPOSE(temp);
END IF;

        mindist := 9999.0; maxdist := 0.0;
        sumTij := 0.0; distAvg := 0.0;
        FOR i := 0 TO nc
          FOR j := i+1 TO nc
            sumTij := sumTij + dist[i][j];
            IF (dist[i][j] < mindist) AND (dist[i][j] > 0.0)
                        mindist := dist[i][j]; END IF;
            IF dist[i][j] > maxdist
                        maxdist := dist[i][j]; END IF;
          END FOR;
        END FOR;
```

```
        distAvg := sumTij / (FLOAT((nc+1)*(nc+1))/2.0 - FLOAT(nc+1) );

        OUTPUT(" ");
        OUTPUT("Average distance to travel is ", distAvg);
        OUTPUT("Min distance to travel is ", mindist);
        OUTPUT("Max distance to travel is ", maxdist);
        str := "Average distance to travel is " + REALTOSTR(distAvg);
        ASK outstrm WriteString(str); ASK outstrm WriteLn;
        str := "Min distance to travel is " + REALTOSTR(mindist);
        ASK outstrm WriteString(str); ASK outstrm WriteLn;
        str := "Max distance to travel is " + REALTOSTR(maxdist);
        ASK outstrm WriteString(str); ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input vehicle's air speed (in mi/hr)");
INPUT(as);

OUTPUT(" ");
OUTPUT("Please input the conversion factor to use with the WIND time matrix");
OUTPUT("The time windows will be updated to ensure the conversion matches");
OUTPUT("  (must be at least as great as previous factor)");
INPUT(windconv);

        {Update tour with windconv to match times}
        FOR i := 0 TO numnodes
                IF i <= nc
                        slo[i] := TRUNC(windconv / factor * FLOAT(slo[i]));
                        shi[i] := TRUNC(windconv / factor * FLOAT(shi[i]));
                END IF;

                tour[i].ea := TRUNC(windconv / factor * FLOAT(tour[i].ea));
                tour[i].la := TRUNC(windconv / factor * FLOAT(tour[i].la));

                IF tour[i].type = 2
                  tour[i].arr := tour[i].ea;
                  tour[i].dep := tour[i].arr;
                END IF;

        END FOR;

str := "Air speed " + REALTOSTR(as);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
str := "Factor used to make the wind time matrix integer" + REALTOSTR(windconv);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
ASK outstrm WriteLn;


OUTPUT(" ");
OUTPUT("Please input the number of tabu search iterations");
OUTPUT("you would like to step through.");
INPUT(iters);

ASK outstrm WriteLn;
str :="# Iters = " + INTTOSTR(iters);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
```

```
OUTPUT(" ");
OUTPUT("Please input the number of days for which you would like to ");
OUTPUT("test random scenarios.");
INPUT(numdays);

OUTPUT(" ");
OUTPUT("Do you want random wind effects on every day");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(windques);

IF windques = 1

  OUTPUT(" ");
  OUTPUT("Input seed number to use for wind mag");
  INPUT(magseed);
  OUTPUT(" ");
  OUTPUT("Input seed number to use for wind dir");
  INPUT(dirseed);

ASK outstrm WriteLn;
str := "magseed="+INTTOSTR(magseed)+" dirseed="+INTTOSTR(dirseed);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

  NEW(randObj1);  ASK randObj1 SetSeed(FetchSeed(magseed));
  NEW(randObj2);  ASK randObj2 SetSeed(FetchSeed(dirseed));

  OUTPUT(" ");
  OUTPUT("Please input the range of DEGREES you would like to test");
  OUTPUT("  - Put lowest number first");
  OUTPUT("  - If testing winds around the 0 deg direction,");
  OUTPUT("    Make sure lowdeg is negative");
  INPUT(lowdeg);
  INPUT(highdeg);

  OUTPUT(" ");
  OUTPUT("Please input the range of MAGNITUDE you would like to test");
  OUTPUT("  - Put lowest number first");
  INPUT(lowmag);
  INPUT(highmag);

ASK outstrm WriteLn;
str :="RANDOM WINDS: degrees " + INTTOSTR(lowdeg) + " " + INTTOSTR(highdeg)
          + " magnitude " + INTTOSTR(lowmag) + " " + INTTOSTR(highmag);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

ELSE

  OUTPUT(" ");
  OUTPUT("Please input the magnitude of the wind vector (in mi/hr)");
  INPUT(wmag);

  OUTPUT(" ");
  OUTPUT("Please input the direction that the wind is blowing FROM in degrees");
```

```
  OUTPUT("  (due EAST is 0 degs, due NORTH is 90 degs, and so on)");
  INPUT(wdir);

  wdir := pi / 180.0 * wdir;

END IF;

OUTPUT(" ");
OUTPUT("Do you want to input the initial tour?");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(startques);

IF startques = 1

  OUTPUT(" ");
  OUTPUT("Input the file from which to read the initial tour");
  INPUT(startfile);

  {open problem file}
  NEW(instrm2);

  filein := startfile + ".DAT";
  ASK instrm2 Open(filein, Input);

  {initialize array of node id's}
  NEW(m, 0..numnodes);
  FOR j := 1 TO nc
          m[j] := 0;
  END FOR;

  ASK instrm2 ReadInt(nvInit);

  IF nvInit <> nv
    OUTPUT("nv and # vehicles in initial tour do not agree -- Break program!!");
  END IF;

  FOR i := 0 TO numnodes
    ASK instrm2 ReadInt(m[i]);  {m contains the id at position i}
  END FOR;

  ASK instrm2 Close;        DISPOSE(instrm2);

  OUTPUT(" ");
  OUTPUT("Do you want to input an initialization set?");
  OUTPUT("  - 1 = YES");
  OUTPUT("  - 0 = NO");
  INPUT(initques);


  IF initques = 0
    totaldays := numdays;
    nInitdays := 0;
    startday := 1;
```

```
ELSE

 OUTPUT(" ");
 OUTPUT("Input the name of the file of tour arrays");
 INPUT(filetour);

 OUTPUT(" ");
 OUTPUT("Input the name of the file of the route frequency matrix");
 INPUT(filefreq);

 NEW(instrm3); NEW(instrm4);

 filetour := filetour + ".DAT";
 ASK instrm3 Open(filetour, Input);

 filefreq := filefreq + ".DAT";
 ASK instrm4 Open(filefreq, Input);

 {Read in the number of tours in the initialization set}
 ASK instrm3 ReadInt(nInitdays);
 totaldays := nInitdays + numdays;
 startday := nInitdays + 1;

 NEW(tourChoice, 1..totaldays, 0..numnodes);

 {Read in the initialization set}
 FOR i := 1 TO nInitdays
  NEW(oldtour, 0..numnodes);

      FOR j := 0 TO numnodes

        NEW(node);
        oldtour[j] := node;

        ASK instrm3 ReadInt(oldtour[j].id);
        {set node types}
        IF (oldtour[j].id = 0) OR (oldtour[j].id > nc)
               oldtour[j].type := 2;          {2=veh node}
        ELSE
          oldtour[j].type := 1;     {1=cust node}
        END IF;

        oldtour[j].ea := tour[oldtour[j].id].ea;
        oldtour[j].la := tour[oldtour[j].id].la;
        oldtour[j].dep := tour[oldtour[j].id].dep;
        oldtour[j].arr := tour[oldtour[j].id].arr;

        tourChoice[i][j] := CLONE(oldtour[j]);

      END FOR;
 END FOR;

OUTPUT("aa");

 {Read in the route frequency matrix of the initialization set}
```

```
    NEW(routefreq, 0..numnodes, 0..numnodes);
    FOR i := 0 TO numnodes
         FOR j := 0 TO numnodes
          ASK instrm4 ReadInt(routefreq[i][j]);
         END FOR;
    END FOR;

    {output route frequency matrix}
    where := "ROUTE FREQUENCY MATRIX, History included ";
    timeToFile(where, outstrm, routefreq, numnodes);

    ASK instrm3 Close; ASK instrm4 Close;
    DISPOSE(instrm3); DISPOSE(instrm4);

   END IF;  {initques}

  END IF;


  OUTPUT(file);
  OUTPUT(filein);
  OUTPUT(fileout);
  OUTPUT(fileout2);

  {*} {denotes a parameter setting}
  {*       nv := 10;              *}
  {*       windconv := 10.0; *}
  {*       iters := 1000;    *}
  {*}      TWPEN := 1.0;
  {*}      gamma := 0;

  {*}      INCREASE := 1.2;
  {*}      DECREASE := 0.9;
  {*}      CYMAX := 50;
  {*}      HTSIZE := 131073;
  {*}      ZRANGE := 1009;
  {*}      minTL := 5;
  {*}      maxTL := 2000;


  {*}      DEPTH := nc+nv-1;
  {*}      tabuLen := MINOF(30, nc+nv-1);

           {**** LOOP OF SCENARIOS ***}

           NEW(windmag, startday..totaldays);
           NEW(winddir, startday..totaldays);
           NEW(duration, startday..totaldays);
           NEW(scores, 1..totaldays);

           NEW(besttype, 1..totaldays);

           IF initques = 1
      FOR i := 1 TO nInitdays
             besttype[i] := 1;  {since only feasible tours used}
```

```
        END FOR;
       END IF;

       IF initques = 0
        NEW(tourChoice, 1..numdays, 0..numnodes);
        NEW(routefreq, 0..numnodes, 0..numnodes);

        {initialize matrix of route frequency counts}
        FOR i := 0 TO numnodes
         FOR j := 0 TO numnodes
               routefreq[i][j] := 0;
         END FOR;
        END FOR;
       END IF;

       NEW(startTour); {find initial tour and/or initial penalties}


{***}  FOR day := startday TO totaldays          {**********}

ASK outstrm WriteLn;
str :="DAY: " + INTTOSTR(day);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

        IF windques = 1
         windmag[day] := ASK randObj1 UniformInt(lowmag, highmag);
         wmag := windmag[day];
         winddir[day] := ASK randObj2 UniformInt(lowdeg, highdeg);
         wdir := FLOAT(winddir[day]);
         wdir := pi / 180.0 * wdir;
        END IF;

ASK outstrm WriteLn;
str :="WIND: magnitude ="+INTTOSTR(wmag)+" direction(rads) = " + REALTOSTR(wdir);
ASK outstrm WriteString(str); ASK outstrm WriteLn; ASK outstrm WriteLn;

        IF servques = 1

          FOR i := 1 TO nc

            loiter := ASK randObj3 UniformReal(0.0, 1.0);

            IF loiter <= ploiter
              s[i] := ASK randObj4 UniformInt(slo[i], shi[i]);
            ELSE
              s[i] := slo[i];
            END IF;

          END FOR;

        END IF;

        ASK timeMatrix timeMatrix(nc, numnodes, gamma, as, wmag, wdir, windconv,
                                  coord, s, dist, time, outstrm, startprint);
OUTPUT("b");
```

```
IF day = startday {must find a true initial tour}

  IF startques = 1           {read in initial tour}

        NEW(inittour, 0..numnodes);

        {Reorder tour, currently in numerical order, by the initial tour
         and place temporarily into inittour}

        FOR i := 0 TO numnodes
         inittour[i] := CLONE(tour[m[i]]);
        END FOR;

        {Copy inittour into tour}
        FOR i := 0 TO numnodes
         tour[i] := CLONE(inittour[i]);
        END FOR;

        DISPOSE(inittour);

        tourSched(1, nc, numnodes, tour, time, tourLen, outstrm);

        startTime := SystemTime();

  ELSE

    ASK startTour startTour(nv, nc, time, tour, tourLen,
                          totPenalty, tourhv, startTime, m, outstrm);
  END IF;

  DISPOSE(m);

ELSE

  IF initques = 0

    {lose old tour, use previous days Choice}
    DISPOSE(tour);
    NEW(tour, 0.. numnodes);
    {use the best result of the previous day}
FOR i := 0 TO numnodes
 tour[i] := CLONE(tourChoice[day-1][i]);
END FOR;

  ELSE
OUTPUT("c");
        {use the robust tour, But it may not have TW info with it}
        NEW(inittour, 0..numnodes);

        {So reorder the tour of the previous day into inittour from the robust tour}

        FOR i := 0 TO numnodes
         inittour[i] := CLONE(tour[tourChoice[robustChoice][i].id]);
        END FOR;
```

```
                {Copy inittour into tour}
                FOR i := 0 TO numnodes
                  tour[i] := CLONE(inittour[i]);
                END FOR;

                DISPOSE(inittour);
            END IF;

            {Compute initial schedule, return tour's total travel + wait time}
            tourSched(1, nc, numnodes, tour, time, tourLen, outstrm);

            startTime := SystemTime();

        END IF;
OUTPUT("d");
{*IF startprint*}
ASK outstrm WriteString("startTour complete"); ASK outstrm WriteLn;
qcktourFile(outstrm, tour, numnodes);
{*END IF;*}
            ASK startTour startPenBest(numnodes, tvl, tourLen, tour, TWPEN,
                                totPenalty, penTrav, tourCost, tourPen,
                                bfiter, bfCost, bfTT, bfnv, bestiter,
                                bestCost, bestTT, bestnv, bestTimeF,
                                bestTime, bestTour, bfTour);
OUTPUT("e");
            NEW(rts);
            {conduct RTS}
            ASK rts search(TWPEN, INCREASE, DECREASE, HTSIZE, CYMAX, ZRANGE, DEPTH,
                           minTL, maxTL, tabuLen, iters, nc, numnodes,
                           outstrm, outstrm2, tourPen, time, stepprint,
                           moveprint, cycleprint, tourCost, penTrav,
                           totPenalty, tvl, bfCost, bfTT, bfnv, bfiter,
                           bestCost, bestTT, bestnv, bestTime, bestTimeF,
                           bestiter, numfeas, tour, bestTour, bfTour);
            DISPOSE(rts);

            stopTime := SystemTime();

            IF bfiter > -1

             {save the best feasible tour found}
             FOR i := 0 TO numnodes
               tourChoice[day][i] := CLONE(bfTour[i]);
             END FOR;

             {output the results}
             where := "DAY " + INTTOSTR(day) + "  BEST FEASIBLE TOUR";
             twServToFile(where, outstrm, bfTour, nc, numnodes, bfCost,
                     windconv, loadprint, s, slo, shi);

             duration[day] := bestTimeF - startTime;
             besttype[day] := 1;

             ASK outstrm WriteString("# vehicles used = ");
```

```
          ASK outstrm WriteInt(bfnv, 2); ASK outstrm WriteLn;

          ASK outstrm WriteString("Best Feasible solution found after ");
          ASK outstrm WriteString(INTTOSTR(bestTimeF-startTime)+" secs");
          ASK outstrm WriteLn;
          ASK outstrm WriteString("on Iteration: "+ INTTOSTR(bfiter));
          ASK outstrm WriteLn;
          ASK outstrm WriteString("with travel time = "+ INTTOSTR(bfTT));
          ASK outstrm WriteLn;

          {update the route frequency matrix}
          FOR i := 0 TO numnodes-1
               j := bfTour[i].id;
               k := bfTour[i+1].id;
               routefreq[j][k] := routefreq[j][k] + 1;
          END FOR;

OUTPUT("f");
         ELSE

          {save the best tour found}
          FOR i := 0 TO numnodes
            tourChoice[day][i] := CLONE(bestTour[i]);
          END FOR;

          {output the results}
          where := "DAY " + INTTOSTR(day)
                         + "  Search complete: BEST TOUR (NOT FEASIBLE)";
          twServToFile(where, outstrm, bestTour, nc, numnodes, bestCost,
                  windconv, loadprint, s, slo, shi);

          duration[day] := bestTime - startTime;
          besttype[day] := 0;

          ASK outstrm WriteString("# vehicles used = ");
          ASK outstrm WriteInt(bestnv, 2); ASK outstrm WriteLn;
          ASK outstrm WriteString("Best solution found after ");
          ASK outstrm WriteString(INTTOSTR(bestTime-startTime)+" secs");
          ASK outstrm WriteLn;
          ASK outstrm WriteString("on Iteration: "+ INTTOSTR(bestiter));
          ASK outstrm WriteLn;
          ASK outstrm WriteString("with travel time = "+ INTTOSTR(bestTT));
          ASK outstrm WriteLn;
{** DONT UPDATE FREQ MATRIX WITH BAD TOUR
          {update the route frequency matrix}
          FOR i := 0 TO numnodes-1
               j := bestTour[i].id;
               k := bestTour[i+1].id;
               routefreq[j][k] := routefreq[j][k] + 1;
          END FOR;
****}
         END IF;

         {Output service time difference}
         servSum := 0.0;
```

```
FOR i := 1 TO nc
 servSum := servSum + FLOAT(s[i] - slo[i]);
END FOR;
servSum := servSum / windconv;
str := "Sum of increase over min service times = "+ REALTOSTR(servSum);
ASK outstrm WriteLn; ASK outstrm WriteString(str);
ASK outstrm WriteLn;

{If we're in the test set, find the Robust Tour every day}
IF initques = 1
        {find most robust tour chosen}
        robustChoice := 1;
        dayscore := 0;
        maxdayscore := 0;
        sumScores := 0;

        FOR rday := 1 TO day

          FOR i := 0 TO numnodes-1

            dayscore := dayscore
      + routefreq[tourChoice[rday][i].id][tourChoice[rday][i+1].id];
          END FOR;
          scores[rday] := dayscore;
          sumScores := sumScores + dayscore;

          {choose the tour with the most robust routes}
          IF dayscore > maxdayscore
            robustChoice := rday;
            bestscore := dayscore;
            maxdayscore := dayscore;

          ELSIF dayscore = maxdayscore

            {choose feasible tours over nonfeas, or choose the most recent}
            IF besttype[rday] >= besttype[robustChoice]
              robustChoice := rday;
              bestscore := dayscore;
            END IF;

          END IF;

          dayscore := 0;

        END FOR; {rday}

        str := "Day = "+INTTOSTR(day)+" and Robust Choice = "
                 +INTTOSTR(robustChoice);
        ASK outstrm WriteLn; ASK outstrm WriteString(str);
        ASK outstrm WriteLn;

END IF;

{output coords to file so we can scatter plot tours}
where := "Day = " + INTTOSTR(day);
```

```
      LatLongToFile(where, outstrm2, tourChoice[day], nc, numnodes, coord);

      DISPOSE(bfTour);
      DISPOSE(bestTour);
      DISPOSE(tourPen);

    END FOR;
    {**** END OF DAY LOOP ****}
OUTPUT("k");

    {output route frequency matrix}
    where := "SCENARIO LOOP COMPLETE, Frequency of Routes Chosen: ";
    timeToFile(where, outstrm, routefreq, numnodes);

    {find most robust tour chosen}
    dayscore := 0;
    maxdayscore := 0;
    sumScores := 0;
    robustChoice := 1;

    FOR day := 1 TO totaldays

      FOR i := 0 TO numnodes-1
        dayscore := dayscore
              + routefreq[tourChoice[day][i].id][tourChoice[day][i+1].id];
      END FOR;
      scores[day] := dayscore;
      sumScores := sumScores + dayscore;

      {choose the tour with the most robust routes}
      IF dayscore > maxdayscore
        robustChoice := day;
        bestscore := dayscore;
        maxdayscore := dayscore;

      ELSIF dayscore = maxdayscore

        {choose feasible tours over nonfeas, or choose the most recent}
        IF besttype[day] >= besttype[robustChoice]
          robustChoice := day;
          bestscore := dayscore;
        END IF;

      END IF;

      dayscore := 0;

    END FOR;  {DAY LOOP}

OUTPUT("1");
    {output robust tour}
    tourSched(1, nc, numnodes, tourChoice[robustChoice], time, tourLen, outstrm);
    countVeh(numnodes, tourChoice[robustChoice], nvu);

    where := "MOST ROBUST TOUR: day = " + INTTOSTR(robustChoice);
```

```
	twServToFile(where, outstrm, tourChoice[robustChoice],
		nc, numnodes, tourLen, windconv, loadprint, s, slo, shi);

	ASK outstrm WriteString("# vehicles used = ");
	ASK outstrm WriteInt(nvu, 2); ASK outstrm WriteLn;
	ASK outstrm WriteString("With robustness score "+ INTTOSTR(bestscore));
	ASK outstrm WriteLn; ASK outstrm WriteLn;
OUTPUT("2");
	ASK outstrm WriteString("MEAN robustness score "
			+ INTTOSTR(sumScores DIV totaldays));
	ASK outstrm WriteLn; ASK outstrm WriteLn;

	{Output Robustness scores}
	ASK outstrm WriteLn;
	ASK outstrm WriteString("Robustness scores: ");
	FOR i := 1 TO totaldays
	  ASK outstrm WriteInt(i, 3);
	  ASK outstrm WriteInt(scores[i], 5);
	  ASK outstrm WriteLn;
	END FOR;

	{Output Robust tour for future Initial tour}
	ASK outInit WriteInt(nv, 3); ASK outInit WriteLn;
	FOR i:= 0 TO numnodes
	  ASK outInit WriteInt(tourChoice[robustChoice][i].id, 5);
	  ASK outInit WriteLn;
	END FOR;

	ASK outstrm Close;
	ASK outInit Close;
    ASK outstrm2 Close;

	DISPOSE(startTour);
	DISPOSE(timeMatrix);
	DISPOSE(outstrm);
	DISPOSE(outInit);
	DISPOSE(outstrm2);
	DISPOSE(s);
	DISPOSE(time);
	DISPOSE(coord);

	IF windques = 1
	DISPOSE(randObj1); DISPOSE(randObj2);
	END IF;
	IF servques = 1
	DISPOSE(randObj3); DISPOSE(randObj4);
	END IF;

END MODULE; {MAIN}
```

Appendix J: MuavEval

A second step to MuavThreat2, the main module MuavEval runs the *evaluation phase* of UAV problems with stochastic winds, service times, and threats.

```
MAIN MODULE uavEval;

FROM IOMod IMPORT StreamObj, ALL FileUseType, ReadKey;
FROM OSMod IMPORT SystemTime;
FROM MathMod IMPORT pi;

FROM uavMod IMPORT timeMatrixObj;
FROM twReduceMod IMPORT twReductionObj;
FROM uavMod IMPORT startUAVObj;
FROM uavMod IMPORT uavRTSobj;

FROM tabuMod IMPORT arrReal2dimType;
FROM tabuMod IMPORT coordArrType;
FROM tabuMod IMPORT tourType;
FROM tabuMod IMPORT nodeType;
FROM tabuMod IMPORT vrpPenType;
FROM tabuMod IMPORT arrInt2dimType;
FROM tabuMod IMPORT arrIntType;
FROM tabuMod IMPORT arrRealType;

FROM tabuMod IMPORT SwapNode;
FROM uavMod IMPORT twCvrgServToFile;
FROM tabuMod IMPORT LatLongToFile;
FROM tabuMod IMPORT qcktourFile;
FROM tabuMod IMPORT tourToScreen;
FROM tabuMod IMPORT timeToFile;
FROM tabuMod IMPORT tourSched;
FROM tabuMod IMPORT countVeh;

FROM uavMod IMPORT expCvrg;

FROM RandMod IMPORT RandomObj, SetSeed, FetchSeed;

VAR
        timeMatrix : timeMatrixObj;
        twReduce : twReductionObj;
        startTour : startUAVObj;
        rts : uavRTSobj;
        randObj1, randObj2, randObj3, randObj4, randObj5 : RandomObj;

        instrm,
        instrm2, instrm3, instrm4,
        outstrm,
        outstrm2,
        outInit : StreamObj;
```

```
factor,    {used to convert the time windows to integer values}
TWPEN,          {Penalty weight assigned to the sum of late arr TW violations}
INCREASE,       {RTS parameter: mult. factor to decrease tabu length}
DECREASE,       {RTS parameter: mult. factor to increase tabu length}

windconv,   {multiplied by the resulting UAV time matrix, it provides an
            integer matrix (for calc speed) with the needed precision}
sumTij,         {sum of the i to j distances in the distance matrix}
mindist, {minimum travel distance}
maxdist, {maximum travel distance}
distAvg, {avg travel distance}
wdir,           {direction of wind vector}
ploiter,  {probabilitiy you loiter over a target}
loiter,         {loiter? - individual node result}
riskadj,  {amount to randomly adjust a target's prob of survival}
PSFCT,          {factor multiplied by coverage results to get more info into
                 the integer move value}
cvrg,           {expected coverage of the tour}
bfCvrg,  {exp coverages of best and best feas tours}
bestCvrg,
servSum         {sum of increase over minimum service times}

        : REAL;

i, j, k,
endnum,         {end number in a numbered data file group}
maxtime,        {max possible time of arrival to any node, for time read}
numcycles,      {number of TW reduction cycles wanted}
numchanges,     {number of TWs reduced by TW reduction Obj}
numnodes,       {number of nodes in the directed graph}
nv,             {number of vehicles}
nc,             {number of targets/customers}
gamma,          {arbitrary cost assigned to the use of each vehicle}
iters,          {number of Tabu Search Iterations per problem}
tourLen, {Length of tour in time}
tvl,            {travel time of tour}
totPenalty,     {Total Penalty assigned to current tour}
tourCost,       {tour Length + Time Window Cost}
penTrav,        {tourCost - totWait == travel time + TW penalty}
bfCost,         {lowest tourCost found for a feasible tour}
bestCost,       {lowest tourCost found for a any tour}
bestTT,         {lowest travel time found for a any tour}
bestnv,         {# vehs used by best overall tour}
bfTT,           {lowest travel time found for feasible tour}
bfnv,           {# vehs used by best feas tour}
bfiter,    {iteration # when best feasible tour found}
tourhv,         {tour's hashing value}
bestiter, {iteration the best Tour found}
bestTime,       {Time the best Tour found}
bfTime,         {Time the best feasible Tour found}
numfeas,        {number of feasible solns found}
startTime,      {start Time (after time matrix, before TW reductions)}
stopTime,       {stop Time (after last iteration)}

DEPTH,          {depth of nodes we look for insert moves}
```

```
ZRANGE,              {upper bound on random integer weights assigned to nodes}
HTSIZE,              {size of hash table array}
CYMAX,               {max cyleLength used to alter mavg}
tabuLen, {current length of tabu tenure}
minTL,          {minimum Tabu Length}
maxTL,          {maximum Tabu Length}

wmag,           {magnitude of wind vector}
as,             {UAV's air speed}
numdays,        {number of days to run random scenarios}
day,            {index of current day}
nInitdays,      {number of days in the initialization set}
totaldays,      {nInitdays + numdays}
nvInit,         {# vehicles in initial tour read from a file}
nvu,            {# vehicles used in current tour}

windques,       {ask whether or not you want random winds}
magseed, dirseed,         {seeds for random winds}
startques,      {ask whether or not you want to input the initial tour}
servseed, loitseed,       {seeds for random service times}
servques,       {ask whether or not you want random service times}
initques, {ask if an initialization set already performed}
riskques,       {ask if random service times are needed}
cvrseed,

lowdeg,         {low end of range of wind direction to test}
highdeg, {high end of range of wind direction to test}
lowmag,         {low end of range of wind magnitude to test}
highmag,        {high end of range of wind magnitude to test}
minloiter,
maxloiter,      {minimum & maximum loiter time}

dayscore,       {robustness score of day under consideration}
maxdayscore,    {max robustness measure found}
bestscore,      {robustness measure of best route found}
sumScores,      {sum of all dayscores, used to find a mean}
robustChoice,   {tags the resulting tour chosen as most robust}

startday, {# of day beginning the scenario, day, loop}
rday     {loop var of the incremental robust tour choice}
         : INTEGER;

outfile,  {name of output file}
where,          {where in the code?}
str,

startfile,
file, filein,   {filenames}
filebegin,
fileout3,
fileout2,
filetour, filefreq,
fileout : STRING;

loadprint,           {print load on vehicles}
```

```
stepprint,                  {print each move evaluation}
moveprint,                  {print every insert move made by RTS}
startprint,                 {print starting tour and tw reduction steps}
cycleprint,                 {print hash results}
timeprint,                  {print time matrix}
twrdprint : BOOLEAN;        {print tw reduction steps}

psurv      : arrRealType;          {prob of survival array}

coord      : coordArrType;                {coordinates array}

bfTour,                                   {best feasible tour found}
bestTour,                                 {node array holding best tour}
tour,                                     {node array holding the tour}
oldtour,                           {temporary tour}
inittour   : tourType;                    {tour to read in an initial tour}

tourPen    : vrpPenType;           {record of curr tour penalties}

windmag,                           {array of wind magnitude per day}
winddir,                    {array of wind direction per day}
duration,                   {array of time to best solution per day}
besttype,                   {array tracking type of best: 1=feas, 0=not}
scores,                            {array of robustness scores}

m,                                 {array of TW midpoints}
slo, shi,                   {arrays of service time ranges}
s        : arrIntType;      {service times used}

dist       : arrReal2dimType;             {no wind distance matrix}

temp,

routefreq,                         {counts the frequency that route i to j
                                    is chosen, where i and j are the array
                                    indices, in that order}
time       : arrInt2dimType;              {time matrix}

tourChoice : ARRAY INTEGER OF tourType; {array of tour choices per day}
psurvDay : ARRAY INTEGER OF arrRealType; {array of psurv per day}

node : nodeType;


BEGIN

{INITIALIZE}
startprint := FALSE;        {print starting tour}
timeprint := FALSE;         {print time matrix}
stepprint := FALSE;         {print each RTS step eval}
moveprint := FALSE;         {print each RTS insert move}
twrdprint := FALSE;         {print TW reduction steps}
cycleprint := FALSE;        {print cycle/nocycle steps}
loadprint := FALSE;         {print quantity & vehicle loads}
```

```
OUTPUT(" ");
OUTPUT("Please input the problem to work on:");
INPUT(file);

        NEW(instrm);                                    {open problem file}
        NEW(outstrm);                                   {open results file}
        NEW(outInit);                         {open file for future initial tour}
        filein := file + ".DAT";
        fileout := file + ".OUT";
        fileout2 := file + "Init.OUT";
        ASK instrm Open(filein, Input);
        ASK outstrm Open(fileout, Output);
        ASK outInit Open(fileout2, Output);

        fileout3 := file + "Rslt" + ".OUT";
        NEW(outstrm2);
        ASK outstrm2 Open(fileout3, Output);

str := "FILE: " + file;
ASK outstrm WriteString(str); ASK outstrm WriteLn; ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input the factor (such as 1, 10, 100, etc.) necessary to convert");
OUTPUT("the time window info to integer quantities");
INPUT(factor);

str := "Factor used for target windows and distances " + REALTOSTR(factor);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input the number of vehicles");
INPUT(nv);

        NEW(timeMatrix);


OUTPUT(" ");
OUTPUT("Do you want random service times?");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(servques);

IF servques = 1

  OUTPUT(" ");
  OUTPUT("Input seed number to use for service time randomization");
  INPUT(servseed);
  OUTPUT(" ");
  OUTPUT("Input seed number to use for loiter randomization");
  INPUT(loitseed);

  NEW(randObj3); NEW(randObj4);
  ASK randObj3 SetSeed(FetchSeed(loitseed));
  ASK randObj4 SetSeed(FetchSeed(servseed));
```

```
 OUTPUT(" ");
 OUTPUT("Give the probability you will loiter over a target");
 INPUT(ploiter);

ASK outstrm WriteLn;
str := "loitseed="+INTTOSTR(loitseed)+" servseed="+INTTOSTR(servseed)+
      "  Pr{loiter} = "+REALTOSTR(ploiter);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

        {Reads in a coords in miles scenario with Service time ranges and psurv}
        ASK timeMatrix readUAVloiter(instrm, nc, numnodes, factor, nv,
                                        psurv, coord, tour, slo, shi,
                                        outstrm, startprint);

        NEW(s, 0..nc);
        s[0] := 0;

ELSE

        {reads UAV file, finds nc, inits coord & tour}
        ASK timeMatrix readUAV(instrm, nc, numnodes, factor, nv,
                                psurv, coord, tour, s, outstrm,
                                    startprint);


END IF;

        ASK instrm Close;          DISPOSE(instrm);

        {compute distance matrix, given coordinates in miles}
        ASK timeMatrix distMatrix(nc, numnodes, coord, dist, outstrm);

IF timeprint
{output distance matrix}
NEW(temp, 0..numnodes, 0..numnodes);
where := "No wind distance Matrix complete";
FOR i := 0 TO numnodes
  FOR j := i+1 TO numnodes
    temp[i][j] := TRUNC(dist[i][j]);
    temp[j][i] := temp[i][j];
  END FOR;
END FOR;
timeToFile(where, outstrm, temp, numnodes);
DISPOSE(temp);
END IF;

        mindist := 9999.0; maxdist := 0.0;
        sumTij := 0.0; distAvg := 0.0;
        FOR i := 0 TO nc
          FOR j := i+1 TO nc
            sumTij := sumTij + dist[i][j];
            IF (dist[i][j] < mindist) AND (dist[i][j] > 0.0)
                        mindist := dist[i][j]; END IF;
            IF dist[i][j] > maxdist
                        maxdist := dist[i][j]; END IF;
```

```
            END FOR;
          END FOR;

          distAvg := sumTij / (FLOAT((nc+1)*(nc+1))/2.0 - FLOAT(nc+1) );

          OUTPUT(" ");
          OUTPUT("Average distance to travel is ", distAvg);
          OUTPUT("Min distance to travel is ", mindist);
          OUTPUT("Max distance to travel is ", maxdist);
          str := "Average distance to travel is " + REALTOSTR(distAvg);
          ASK outstrm WriteString(str); ASK outstrm WriteLn;
          str := "Min distance to travel is " + REALTOSTR(mindist);
          ASK outstrm WriteString(str); ASK outstrm WriteLn;
          str := "Max distance to travel is " + REALTOSTR(maxdist);
          ASK outstrm WriteString(str); ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input vehicle's air speed (in mi/hr)");
INPUT(as);

OUTPUT(" ");
OUTPUT("Please input the conversion factor to use with the WIND time matrix");
OUTPUT("The time windows will be updated to ensure the conversion matches");
OUTPUT("  (must be at least as great as previous factor)");
INPUT(windconv);

          {Update tour with windconv to match times}
          FOR i := 0 TO numnodes
                    IF i <= nc
                              slo[i] := TRUNC(windconv / factor * FLOAT(slo[i]));
                              shi[i] := TRUNC(windconv / factor * FLOAT(shi[i]));
                    END IF;

                    tour[i].ea := TRUNC(windconv / factor * FLOAT(tour[i].ea));
                    tour[i].la := TRUNC(windconv / factor * FLOAT(tour[i].la));

                    IF tour[i].type = 2
                      tour[i].arr := tour[i].ea;
                      tour[i].dep := tour[i].arr;
                    END IF;

          END FOR;

str := "Air speed " + REALTOSTR(as);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
str := "Factor used to make the wind time matrix integer" + REALTOSTR(windconv);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
ASK outstrm WriteLn;


OUTPUT(" ");
OUTPUT("Please input the number of tabu search iterations");
OUTPUT("you would like to step through.");
INPUT(iters);
```

```
ASK outstrm WriteLn;
str :="# Iters = " + INTTOSTR(iters);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input the number of days for which you would like to ");
OUTPUT("test random scenarios.");
INPUT(numdays);

OUTPUT(" ");
OUTPUT("Do you want random wind effects on every day");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(windques);

IF windques = 1

  OUTPUT(" ");
  OUTPUT("Input seed number to use for wind mag");
  INPUT(magseed);
  OUTPUT(" ");
  OUTPUT("Input seed number to use for wind dir");
  INPUT(dirseed);

ASK outstrm WriteLn;
str := "magseed="+INTTOSTR(magseed)+" dirseed="+INTTOSTR(dirseed);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

  NEW(randObj1);  ASK randObj1 SetSeed(FetchSeed(magseed));
  NEW(randObj2);  ASK randObj2 SetSeed(FetchSeed(dirseed));

  OUTPUT(" ");
  OUTPUT("Please input the range of DEGREES you would like to test");
  OUTPUT("  - Put lowest number first");
  OUTPUT("  - If testing winds around the 0 deg direction,");
  OUTPUT("    Make sure lowdeg is negative");
  INPUT(lowdeg);
  INPUT(highdeg);

  OUTPUT(" ");
  OUTPUT("Please input the range of MAGNITUDE you would like to test");
  OUTPUT("  - Put lowest number first");
  INPUT(lowmag);
  INPUT(highmag);

ASK outstrm WriteLn;
str :="RANDOM WINDS: degrees " + INTTOSTR(lowdeg) + " " + INTTOSTR(highdeg)
         + " magnitude " + INTTOSTR(lowmag) + " " + INTTOSTR(highmag);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

ELSE

  OUTPUT(" ");
  OUTPUT("Please input the magnitude of the wind vector (in mi/hr)");
  INPUT(wmag);
```

```
 OUTPUT(" ");
 OUTPUT("Please input the direction that the wind is blowing FROM in degrees");
 OUTPUT("  (due EAST is 0 degs, due NORTH is 90 degs, and so on)");
 INPUT(wdir);

 wdir := pi / 180.0 * wdir;

END IF;

OUTPUT(" ");
OUTPUT("Do you want to input the initial tour?");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(startques);

IF startques = 1

 OUTPUT(" ");
 OUTPUT("Input the file from which to read the initial tour");
 INPUT(startfile);

 {open problem file}
 NEW(instrm2);

 filein := startfile + ".DAT";
 ASK instrm2 Open(filein, Input);

 {initialize array of node id's}
 NEW(m, 0..numnodes);
 FOR j := 1 TO nc
        m[j] := 0;
 END FOR;

 ASK instrm2 ReadInt(nvInit);

 IF nvInit <> nv
   OUTPUT("nv and # vehicles in initial tour do not agree -- Break program!!");
 END IF;

 FOR i := 0 TO numnodes
   ASK instrm2 ReadInt(m[i]);  {m contains the id at position i}
 END FOR;

 ASK instrm2 Close;       DISPOSE(instrm2);

 OUTPUT(" ");
 OUTPUT("Do you want to input an initialization set?");
 OUTPUT("  - 1 = YES");
 OUTPUT("  - 0 = NO");
 INPUT(initques);


 IF initques = 0
  totaldays := numdays;
```

```
  nInitdays := 0;
  startday := 1;

ELSE

  OUTPUT(" ");
  OUTPUT("Input the name of the file of tour arrays");
  INPUT(filetour);

  OUTPUT(" ");
  OUTPUT("Input the name of the file of the route frequency matrix");
  INPUT(filefreq);

  NEW(instrm3); NEW(instrm4);

  filetour := filetour + ".DAT";
  ASK instrm3 Open(filetour, Input);

  filefreq := filefreq + ".DAT";
  ASK instrm4 Open(filefreq, Input);

  {Read in the number of tours in the initialization set}
  ASK instrm3 ReadInt(nInitdays);
  totaldays := nInitdays + numdays;
  startday := nInitdays + 1;

  NEW(tourChoice, 1..totaldays, 0..numnodes);

  {Read in the initialization set}
  FOR i := 1 TO nInitdays
   NEW(oldtour, 0..numnodes);

       FOR j := 0 TO numnodes

         NEW(node);
         oldtour[j] := node;

         ASK instrm3 ReadInt(oldtour[j].id);
         {set node types}
         IF (oldtour[j].id = 0) OR (oldtour[j].id > nc)
               oldtour[j].type := 2;          {2=veh node}
         ELSE
           oldtour[j].type := 1;     {1=cust node}
         END IF;

         oldtour[j].ea := tour[oldtour[j].id].ea;
         oldtour[j].la := tour[oldtour[j].id].la;
         oldtour[j].dep := tour[oldtour[j].id].dep;
         oldtour[j].arr := tour[oldtour[j].id].arr;

         tourChoice[i][j] := CLONE(oldtour[j]);

       END FOR;
  END FOR;
```

```
OUTPUT("aa");

   {Read in the route frequency matrix of the initialization set}
   NEW(routefreq, 0..numnodes, 0..numnodes);
   FOR i := 0 TO numnodes
          FOR j := 0 TO numnodes
            ASK instrm4 ReadInt(routefreq[i][j]);
          END FOR;
   END FOR;

   {output route frequency matrix}
   where := "ROUTE FREQUENCY MATRIX, History included ";
   timeToFile(where, outstrm, routefreq, numnodes);

   ASK instrm3 Close; ASK instrm4 Close;
   DISPOSE(instrm3); DISPOSE(instrm4);

 END IF;  {initques}

END IF;


OUTPUT(" ");
OUTPUT("Do you want random THREAT effects on every day");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(riskques);

OUTPUT(" ");
OUTPUT("Input the factor to convert coverage to an integer value");
INPUT(PSFCT);

IF riskques = 1

 OUTPUT(" ");
 OUTPUT("Input seed number to use for random COVERAGES");
 INPUT(cvrseed);

ASK outstrm WriteLn;
str := "RANDOM THREATS: cvrseed="+INTTOSTR(cvrseed);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

 NEW(randObj5);  ASK randObj5 SetSeed(FetchSeed(cvrseed));

END IF;


OUTPUT(file);
OUTPUT(filein);
OUTPUT(fileout);
OUTPUT(fileout2);

{*} {denotes a parameter setting}
{*       nv := 10;              *}
{*       windconv := 10.0; *}
```

```
{*      iters := 1000;   *}
{*}     TWPEN := 10.0;
{*}     gamma := 0;

{*}     INCREASE := 1.2;
{*}     DECREASE := 0.9;
{*}     CYMAX := 50;
{*}     HTSIZE := 131073;
{*}     ZRANGE := 1009;
{*}     minTL := 5;
{*}     maxTL := 2000;


{*}     DEPTH := nc+nv-1;
{*}     tabuLen := MINOF(30, nc+nv-1);

        {**** LOOP OF SCENARIOS ***}

        NEW(windmag, startday..totaldays);
        NEW(winddir, startday..totaldays);
        NEW(duration, startday..totaldays);
        NEW(psurvDay, startday..totaldays, 0..nc);
        NEW(scores, 1..totaldays);

        NEW(besttype, 1..totaldays);

        IF initques = 1
     FOR i := 1 TO nInitdays
           besttype[i] := 1;  {since only feasible tours used}
         END FOR;
        END IF;

        IF initques = 0
         NEW(tourChoice, 1..numdays, 0..numnodes);
         NEW(routefreq, 0..numnodes, 0..numnodes);

         {initialize matrix of route frequency counts}
         FOR i := 0 TO numnodes
          FOR j := 0 TO numnodes
                routefreq[i][j] := 0;
          END FOR;
         END FOR;
        END IF;

        NEW(startTour); {find initial tour and/or initial penalties}


{***}   FOR day := startday TO totaldays          {**********}

ASK outstrm WriteLn;
str :="DAY: " + INTTOSTR(day);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

          IF windques = 1
           windmag[day] := ASK randObj1 UniformInt(lowmag, highmag);
```

```
      wmag := windmag[day];
      winddir[day] := ASK randObj2 UniformInt(lowdeg, highdeg);
      wdir := FLOAT(winddir[day]);
      wdir := pi / 180.0 * wdir;
    END IF;

ASK outstrm WriteLn;
str :="WIND: magnitude ="+INTTOSTR(wmag)+" direction(rads) = " + REALTOSTR(wdir);
ASK outstrm WriteString(str); ASK outstrm WriteLn; ASK outstrm WriteLn;


    IF riskques = 1

      {randomly adjust the prob of survival at the target nodes}
      FOR i := 1 TO nc
        riskadj := ASK randObj3 UniformReal(-1.0, 1.0);
            IF riskadj < -0.333
                    riskadj := -1.0;
            ELSIF riskadj > 0.333
                    riskadj := 1.0;
    ELSE
                    riskadj := 0.0;
            END IF;
        psurvDay[day][i] := psurv[i] + riskadj / 10.0;

      END FOR;

    ELSE
            psurvDay[day] := psurv;
    END IF;
OUTPUT("aaa");

    IF servques = 1

      FOR i := 1 TO nc

        loiter := ASK randObj3 UniformReal(0.0, 1.0);

        IF loiter <= ploiter
          s[i] := ASK randObj4 UniformInt(slo[i], shi[i]);
        ELSE
          s[i] := slo[i];
        END IF;

      END FOR;

    END IF;

    ASK timeMatrix timeMatrix(nc, numnodes, gamma, as, wmag, wdir, windconv,
                              coord, s, dist, time, outstrm, timeprint);
OUTPUT("b");

    IF day = startday {must find a true initial tour}

      IF startques = 1          {read in initial tour}
```

```
            NEW(inittour, 0..numnodes);

            {Reorder tour, currently in numerical order, by the initial tour
             and place temporarily into inittour}

            FOR i := 0 TO numnodes
              inittour[i] := CLONE(tour[m[i]]);
            END FOR;

            {Copy inittour into tour}
            FOR i := 0 TO numnodes
              tour[i] := CLONE(inittour[i]);
            END FOR;

            DISPOSE(inittour);

            tourSched(1, nc, numnodes, tour, time, tourLen, outstrm);

            startTime := SystemTime();

        ELSE

          ASK startTour startTour(nv, nc, time, tour, tourLen,
                                 totPenalty, tourhv, startTime, m, outstrm);
        END IF;

        DISPOSE(m);

      ELSE

        IF initques = 0

          {lose old tour, use previous days Choice}
          DISPOSE(tour);
          NEW(tour, 0.. numnodes);
          {use the best result of the previous day}
      FOR i := 0 TO numnodes
        tour[i] := CLONE(tourChoice[day-1][i]);
      END FOR;

        ELSE
OUTPUT("c");
            {use the robust tour, But it may not have TW info with it}
            NEW(inittour, 0..numnodes);

            {So reorder the tour of the previous day into inittour from the robust tour}

            FOR i := 0 TO numnodes
              inittour[i] := CLONE(tour[tourChoice[robustChoice][i].id]);
            END FOR;

            {Copy inittour into tour}
            FOR i := 0 TO numnodes
              tour[i] := CLONE(inittour[i]);
```

```
            END FOR;

            DISPOSE(inittour);
        END IF;

        {Compute initial schedule, return tour's total travel + wait time}
        tourSched(1, nc, numnodes, tour, time, tourLen, outstrm);

        startTime := SystemTime();

    END IF;
OUTPUT("d");
IF startprint
{* ASK outstrm WriteString("startTour complete"); ASK outstrm WriteLn;
qcktourFile(outstrm, tour, numnodes); *}
where := "startTour complete";
twCvrgServToFile(where, outstrm, tour, nc, numnodes, tourLen,
            windconv, loadprint, psurv, s, slo);
END IF;
        ASK startTour startUAVbest(numnodes, tvl, tourLen, tour, TWPEN,
                              psurvDay[day], totPenalty, penTrav, tourCost,
                              tourPen, bfiter, bfCost, bfTT, bfnv, bestiter,
                              bestCost, bestTT, bestnv, bfTime,
                              bestTime, cvrg, bfCvrg, bestCvrg,
                              bestTour, bfTour);
OUTPUT("e");
        NEW(rts);
        {conduct RTS}
        ASK rts search(psurvDay[day], PSFCT,
                    TWPEN, INCREASE, DECREASE, HTSIZE, CYMAX, ZRANGE, DEPTH,
                    minTL, maxTL, tabuLen, iters, nc, numnodes,
                    outstrm, outstrm2, tourPen, time, stepprint,
                    moveprint, cycleprint, tourCost, penTrav, totPenalty, tvl,
                    bfCost, bfTT, bfnv, bfiter, bestCost, bestTT, bestnv,
                    bestTime, bfTime, bestiter, numfeas,
                    bfCvrg, bestCvrg, cvrg,
                    tour, bestTour, bfTour);
        DISPOSE(rts);

        stopTime := SystemTime();

        IF bfiter > -1

         {save the best feasible tour found}
         FOR i := 0 TO numnodes
           tourChoice[day][i] := CLONE(bfTour[i]);
         END FOR;

         {output the results}
         where := "DAY " + INTTOSTR(day) + "  BEST FEASIBLE TOUR";
         twCvrgServToFile(where, outstrm, bfTour, nc, numnodes, bfCost,
                              factor, loadprint, psurvDay[day], s, slo);

         duration[day] := bfTime - startTime;
         besttype[day] := 1;
```

```
            ASK outstrm WriteString("# vehicles used = ");
            ASK outstrm WriteInt(bfnv, 2); ASK outstrm WriteLn;

            ASK outstrm WriteString("Best Feasible solution found after ");
            ASK outstrm WriteString(INTTOSTR(bfTime-startTime)+" secs");
            ASK outstrm WriteLn;
            ASK outstrm WriteString("on Iteration: "+ INTTOSTR(bfiter));
            ASK outstrm WriteLn;
            ASK outstrm WriteString("with travel time = "+ INTTOSTR(bfTT));
            ASK outstrm WriteLn;
            ASK outstrm WriteLn;
            ASK outstrm WriteString("& Expected coverage = "+REALTOSTR(bfCvrg));
            ASK outstrm WriteLn;

            {update the route frequency matrix}
            FOR i := 0 TO numnodes-1
                  j := bfTour[i].id;
                  k := bfTour[i+1].id;
                  routefreq[j][k] := routefreq[j][k] + 1;
            END FOR;

OUTPUT("f");
           ELSE

            {save the best tour found}
            FOR i := 0 TO numnodes
              tourChoice[day][i] := CLONE(bestTour[i]);
            END FOR;

            {output the results}
            where := "DAY " + INTTOSTR(day)
                          + "  Search complete: BEST TOUR (NOT FEASIBLE)";
            twCvrgServToFile(where, outstrm, bestTour, nc, numnodes, bestCost,
                     windconv, loadprint, psurvDay[day], s, slo);

            duration[day] := bestTime - startTime;
            besttype[day] := 0;

            ASK outstrm WriteString("# vehicles used = ");
            ASK outstrm WriteInt(bestnv, 2); ASK outstrm WriteLn;
            ASK outstrm WriteString("Best solution found after ");
            ASK outstrm WriteString(INTTOSTR(bestTime-startTime)+" secs");
            ASK outstrm WriteLn;
            ASK outstrm WriteString("on Iteration: "+ INTTOSTR(bestiter));
            ASK outstrm WriteLn;
            ASK outstrm WriteString("with travel time = "+ INTTOSTR(bestTT));
            ASK outstrm WriteLn;
            ASK outstrm WriteLn;
            ASK outstrm WriteString("& Expected coverage = "+REALTOSTR(bestCvrg));
            ASK outstrm WriteLn;
{** DONT UPDATE FREQ MATRIX WITH BAD TOUR
            {update the route frequency matrix}
            FOR i := 0 TO numnodes-1
                  j := bestTour[i].id;
```

```
        k := bestTour[i+1].id;
        routefreq[j][k] := routefreq[j][k] + 1;
    END FOR;
****}
    END IF;

    {Output service time difference}
    servSum := 0.0;
    FOR i := 1 TO nc
     servSum := servSum + FLOAT(s[i] - slo[i]);
    END FOR;
    servSum := servSum / windconv;
    str := "Sum of increase over min service times = "+ REALTOSTR(servSum);
    ASK outstrm WriteLn; ASK outstrm WriteString(str);
    ASK outstrm WriteLn;

    {If we're in the test set, find the Robust Tour every day}
    IF initques = 1
        {find most robust tour chosen}
        robustChoice := 1;
        dayscore := 0;
        maxdayscore := 0;
        sumScores := 0;

        FOR rday := 1 TO day

          FOR i := 0 TO numnodes-1

            dayscore := dayscore
        + routefreq[tourChoice[rday][i].id][tourChoice[rday][i+1].id];
          END FOR;
          scores[rday] := dayscore;
          sumScores := sumScores + dayscore;

          {choose the tour with the most robust routes}
          IF dayscore > maxdayscore
            robustChoice := rday;
            bestscore := dayscore;
            maxdayscore := dayscore;

          ELSIF dayscore = maxdayscore

            {choose feasible tours over nonfeas, or choose the most recent}
            IF besttype[rday] >= besttype[robustChoice]
              robustChoice := rday;
              bestscore := dayscore;
            END IF;

          END IF;

          dayscore := 0;

        END FOR; {rday}

        str := "Day = "+INTTOSTR(day)+" and Robust Choice = "
```

```
                +INTTOSTR(robustChoice);
        ASK outstrm WriteLn; ASK outstrm WriteString(str);
        ASK outstrm WriteLn;

    END IF;

    {output coords to file so we can scatter plot tours}
    where := "Day = " + INTTOSTR(day);
    LatLongToFile(where, outstrm2, tourChoice[day], nc, numnodes, coord);

    DISPOSE(bfTour);
    DISPOSE(bestTour);
    DISPOSE(tourPen);

   END FOR;
   {**** END OF DAY LOOP ****}
OUTPUT("k");

   {output route frequency matrix}
   where := "SCENARIO LOOP COMPLETE, Frequency of Routes Chosen: ";
   timeToFile(where, outstrm, routefreq, numnodes);

   {find most robust tour chosen}
   dayscore := 0;
   maxdayscore := 0;
   sumScores := 0;
   robustChoice := 1;

   FOR day := 1 TO totaldays

    FOR i := 0 TO numnodes-1
     dayscore := dayscore
          + routefreq[tourChoice[day][i].id][tourChoice[day][i+1].id];
    END FOR;
    scores[day] := dayscore;
    sumScores := sumScores + dayscore;

    {choose the tour with the most robust routes}
    IF dayscore > maxdayscore
     robustChoice := day;
     bestscore := dayscore;
     maxdayscore := dayscore;

    ELSIF dayscore = maxdayscore

     {choose feasible tours over nonfeas, or choose the most recent}
     IF besttype[day] >= besttype[robustChoice]
      robustChoice := day;
      bestscore := dayscore;
     END IF;

    END IF;

    dayscore := 0;
```

```
        END FOR;  {DAY LOOP}

OUTPUT("1");
        {output robust tour}
        tourSched(1, nc, numnodes, tourChoice[robustChoice], time, tourLen, outstrm);
        countVeh(numnodes, tourChoice[robustChoice], nvu);
        expCvrg(numnodes, psurv, tourChoice[robustChoice], cvrg);

        where := "MOST ROBUST TOUR: day = " + INTTOSTR(robustChoice);
        twCvrgServToFile(where, outstrm, tourChoice[robustChoice], nc, numnodes,
                tourLen, windconv, loadprint, psurvDay[totaldays], s, slo);

        ASK outstrm WriteString("# vehicles used = ");
        ASK outstrm WriteInt(nvu, 2); ASK outstrm WriteLn;
        ASK outstrm WriteString("With robustness score "+ INTTOSTR(bestscore));
        ASK outstrm WriteLn; ASK outstrm WriteLn;
OUTPUT("2");
        ASK outstrm WriteString("MEAN robustness score "
                        + INTTOSTR(sumScores DIV totaldays));
        ASK outstrm WriteLn; ASK outstrm WriteLn;

        {Output Robustness scores}
        ASK outstrm WriteLn;
        ASK outstrm WriteString("Robustness scores: ");
        FOR i := 1 TO totaldays
          ASK outstrm WriteInt(i, 3);
          ASK outstrm WriteInt(scores[i], 5);
          ASK outstrm WriteLn;
        END FOR;

        {Output Robust tour for future Initial tour}
        ASK outInit WriteInt(nv, 3); ASK outInit WriteLn;
        FOR i:= 0 TO numnodes
          ASK outInit WriteInt(tourChoice[robustChoice][i].id, 5);
          ASK outInit WriteLn;
        END FOR;

        ASK outstrm Close;
        ASK outInit Close;
    ASK outstrm2 Close;

        DISPOSE(startTour);
        DISPOSE(timeMatrix);
        DISPOSE(outstrm);
        DISPOSE(outInit);
        DISPOSE(outstrm2);
        DISPOSE(s);
        DISPOSE(time);
        DISPOSE(coord);

        IF windques = 1
        DISPOSE(randObj1); DISPOSE(randObj2);
        END IF;
        IF servques = 1
        DISPOSE(randObj3); DISPOSE(randObj4);
```

```
END IF;

END MODULE; {MAIN}
```

Appendix K: Literature Review

This review concentrates on contemporary works relevant to the class of general vehicle routing problems (GVRP) approached within this thesis and works related to our embedded optimization approach. It is separated into three sections, although some works contribute to more than section. Section K.1 provides a summary of the literature relevant to an understanding of the GVRP and the attacked subset. Section K.2 follows with a survey of works describing tabu search (TS) and some relevant applications of this powerful meta-heuristic. Section K.3 reviews literature exploring the utility and relationships important to the embedded optimization of the TS heuristic within a simulation or other problem solving form. Section K.4 describes the resources used by the author as references for CACI's MODSIM programming lanquage. A number of MODSIM objects created by the researcher form a fundamental part of this study. Section K.5 closes with a few conclusions drawn from the literature review.

K.1. The General Vehicle Routing Problem

In their 1997 survey article of commercial vehicle routing software, Hall and Partyka describe the vehicle routing problem (VRP) as an interdependent collection of four problems. One is the calculation of distances between stops, two is the partitioning of the region into districts of feasible routes, three is usually a traveling salesman problem (TSP) or a TSP with time window constraints (TSPTW), and four is the subsequent crew assignment (Hall and Partyka 1997). Within the collection of articles entitled *Vehicle Routing: Method and Studies* (Golden and Assad 1988), Assad places the general characteristics of routing problems into six categories: nature of demand, information on

demand, vehicle fleet, crew requirements, scheduling requirements, and data requirements. Both descriptions illustrate the need for any routing system to incorporate elements of embedded optimization. While these descriptions capture the industrial application of the VRP, they fall short for most military applications as they do not include a variance in the nature of the operational environment.

In his thesis on unmanned aerial vehicle routing, Sisson created a formulation of a multiple vehicle TSPTW (mTSPTW) that incorporated the probabilities of vehicle attrition due to hostile forces into the objective function (Sisson 1997). No civilian approach considers the possibility of attrition due to a hostile environment. Although the commercial market is competitive, combative behavior remains illegal. Sisson also researched the performance of unmanned aerial vehicles the important influence of wind. Sisson's work is a critical stepping stone for this thesis.

As Hall and Partyka noted, an mTSPTW often comprises a critical portion of any larger VRP. To better understand the relationship of problems within the GVRP family, a number of works were consulted. *The Traveling Salesman Problem* holds a prominent standing amongst the literature available. In Chapter 2 of that collection, R. S. Garfinkel quickly summarizes the TSP's "seductive" qualities and provides five distinctly separate applications. Garfinkel also provides some simple transformations to the TSP, including the introduction of multiple salesman (referred to previously as multiple vehicle) and the relaxation allowing the repetition of cities. Garfinkel continues through the generalization of the TSP as an assignment problem and a minimum spanning tree, as well as providing a standard linear programming formulation. Chapter 12 of *The Traveling Salesman Problem*, written by N. Christofides, is entitled "Vehicle routing."

Christofides provides three formulations, some optimization algorithms, and some heuristics. Nemhauser and Wolsey's text, *Integer and Combinatorial Optimization*, provided an example meant for students of the subject after providing a integer programming formulation of the issues involved.

A more current survey of approaches to the TSP and VRP was accomplished in 1992 by Gilbert Laporte. In the first of two invited reviews for the *European Journal of Operational Research*, Laporte surveys the main exact and heuristic algorithms for the TSP. The second article performs the same function for the VRP. After introducing tabu search, Laporte states the "computational results indicate that the proposed heuristic may be one of the best ever developed for the VRP" (Laporte 1992b).

Although the TSP is classified as NP-hard (Glover 1997; Lenstra 1985), special cases exist that can be solved in polynomial time (Glover 1997). In their 1997 article, Glover and Punnen identify new solvable cases of the TSP. A similar work accomplished by the Optimization Group at the Technical University of Graz surveys known efficiently solved cases. This thesis suggests such cases be sought by any VRP software before a more general algorithm, such as tabu search, is applied.

Finally, Carlton's 1995 dissertation is a critical resource. Carlton surveys the proposed classification schemes of the problems within the GVRP class. He concludes that no prior system gives "a direct approach to GVRP classification to enhance the understanding and exploitation of the relationships among the GVRP problem types" (Carlton 1995). He then proposes a hierarchical taxonomy that classifies GVRP types along those lines. In brief, Carlton first summarizes the GVRP into three "floors." Each

floor includes the following cases and their possible combinations (Carlton's notation is slightly altered for more simplicity without loss of information):

1. SV: Single vehicle.
2. MVH: Multiple homogenous vehicles.
3. $\overline{\text{MVH}}$: Multiple non-homogenous vehicles
4. SD: Single depot.
5. MD: Multiple depots.
6. TW: Time window constraints present.
7. RL: Route length constraints present.

The first floor is the family of TSP problems. With the addition of vehicle capacity constraints, one transitions to the second floor of VRP problems. Precedence constraints cause a transition to third floor of pickup and delivery problems (PDP). As a tangible illustration of his success, this research employs Carlton's taxonomy to enact the object-oriented concepts of inheritance and polymorphism between MODSIM optimization objects.

The MODSIM objects accompanying this thesis correspond to the TSP, the mTSP, the mTSPTW, and the VRPTW. Carlton concentrates much his efforts upon the TSPTW and mTSPTW. Of course, a simple transformation creates TSPTW from the mTSPTW. From his literature review, Carlton concludes the TSPTW is not as "well studied" as the TSP and VRP; "it stands in the gap" (Carlton 1995). Carlton's focus on the TSPTW appears well warranted given the previous synopsis of "real-world" applications of the VRP given by Hall and Partyka. This research uses an adaptation of

Carlton's C programming language code for the VRPTW as a first step in the creation of the MODSIM objects.

Jaillet and Odoni (1988) demonstrate the added complexity of a probabilistic TSP (PTSP) over the TSP. For even a simple heuristic like the nearest-neighbor, they find the computational effort increases by $O(n^2)$ over the deterministic version. Furthermore, their formulation of the PTSP is simple in comparison to the UAV problem since they only consider the probability that customers are not present. Jaillet and Odoni sought the similar objective of "well-behaved" or robust routes, but all of their stochastic programming methods were bound to smaller numbers of customers by the necessary computational effort.

K.2. Tabu Search and Applications Related to the GVRP

As mentioned previously, Laporte's survey of VRP algorithms gave highest marks to tabu search (TS). He based this conclusion upon his own version of TS created with Gendreau and Hertz. In their 1996 article, Kervahut, Garcia, and Rousseau compare the performance of their TS heuristic to that of five other documented heuristics within the well known Solomon datasets. Their TS employed a tabu list of fixed length and infeasible regions were not accessible to the search. Yet, the quality of solutions reached by their version of TS bests all other considered heuristics except one known as GIDEON. A statistical test failed to reject the hypothesis that the tested TS heuristic and GIDEON are equally effective (Kervahut 1996). From the literature, it is apparent TS is a powerful heuristic within the GVRP class.

Given this justification for its use, an investigation of TS becomes imperative. Fred Glover introduced TS in 1986 and his writings form a necessary portion of any review regarding this heuristic. His 1990 article, "Tabu Search: A Tutorial," seemed an obvious place to start. Here, Glover provides guidelines in building a TS heuristic. Guidelines include suggestions for the length of tabu lists, the number of attributes considered, the comparison of attributes, the balancing of diversification and intensification efforts to the aspiration criteria, and the powers of target analysis.

As an application of artificial intelligence, target analysis is the application of empirical results from classes of problems to the problems attempted by your own heuristic. Glover strongly emphasizes to use of target analysis to improve move evaluations as his fifth guideline (Glover 1990a). His sixth guideline suggests the use of a frequency-derived (the frequency of revisited solutions) penalty to encourage diversification, with a possible marrying to restarting the search (Glover 1990a).

Glover had stated one year prior to the "tutorial" article in his "Tabu Search-Part 1" publication, that TS was "still in its infancy" (Glover 1989). The tutorial mostly provided a review of "where TS is, and where it is going" with Glover's guidelines, predictions, and other theoretical discussions for the meta-heuristic. Many of the guidelines were the obvious result of Glover's experience in TS application. Similarly, his predictions were educated guesses of things to come. Glover's "Tabu Search-Part I" and Tabu Search-Part II" are more heavily cited than the tutorial article, as they provide a more straight-forward and step-by-step presentation of TS (Glover 1989; 1990b).

With their reactive tabu search (RTS), Battiti and Tecchiolli provided the important next step suggested by Glover's fifth guideline. Battiti and Tecchiolli present

the reactive tabu search in their 1994 article and show it to be a far more robust procedure than the fixed and strict tabu search heuristics. To illustrate the technique's abilities, the authors apply it to the optimization of a quadratic assignment problem and a complex sinusoidal function. The applications are both successful and the authors are kind enough to provide detailed pseudocode. (Battiti 1994)

When one considers that the TS heuristics given by Laporte and Kervahut fall in the category of fixed TS and the RTS achieves significant jumps in computational efficiency without applying Glover's time consuming target analysis preprocessing, it beccomes apparent that Battiti and Tecchiolli have ushered in a powerful improvement to the application of TS. In his 1996 article, Battiti states that "parameter tuning" and "search confinement" are "potential drawbacks to simple implementations" of tabu search, genetic algorithms, and simulated annealing. He then explains that the reactive tabu search effectively overcomes these drawbacks. He classifies the reactive TS as a deterministic algorithm. However, it is quickly made stochastic through random tie breaking and a random "escape trajectory." (Battiti 1996)

Battiti shares the important find that the order of operations required for memory usage per iteration is O(1) or essentially a constant that is independent of the number of iterations performed. He also claims hashing techniques are available that need only a few bytes of memory and result in small "collision" probabilities. He does not support the claim, but he states small collision probabilities do not have statistically significant effects on the reactive TS heuristic. (Battiti 1996)

With a four 0/1 bit example, Battiti then illustrates the properties of reactive TS. In the example, the tabu list length takes progressively more iterations to

increase after the previous increment, and the distance of the search from the optimal "attractor" varies quickly and increases quickly to the maximum range. These illustrations are convincing evidence of the robust balance of intensification and diversification achieved with the reactive TS heuristic. (Battiti 1996)

Once again, Carlton's dissertation is helpful. He provides a two-level open hashing structure. This structure allows the TS to "efficiently store and accurately identify solutions in order to determine whether a particular solution has been visited previously during the search" (Carlton 1995). It minimizes the probability of two non-identical tours being incorrectly determined as identical (Carlton 1995).

It should be noted that Carlton's RTS is deterministic, while the RTS proposed by Battiti and Tecchiolli is stochastic. Carlton's RTS begins from a deterministic starting solution and uses deterministic escape routines, while the heuristic of Battiti and Tecchiolli begins from a stochastic starting solution and uses stochastic escape routines (Carlton 1995).

Although two and a half years have passed since the introduction of RTS, it remains at the forefront of TS heuristics proposed. A contrasting example is provided by Rochat and Taillard. In their 1995 article, the authors propose a technique to overcome the weaknesses of previous local searches and tabu search. The first weakness is the probability of becoming trapped in local optimum, the second is the large computational effort. The approach has two phases. The first begins with an initialization set generated from "good" heuristic solutions. The second phase seeks to extract good tours from the initial set (or from any previous tours after the 2nd iteration) and then seeks to improve this set. The improvement often arises from a combination of the previous "good" tours.

How these "good" solutions are achieved is not specified and appears to be a major weakness of the approach.

As it does with Glover's target analysis, RTS seems to obviate the need for any esoteric pre-processing of the sort used by Rochat and Taillard. These pre-processing techniques report impressive results, but they require a high computational cost and would be difficult to implement by non-TS experts working in the field vehicle routing, civilian and otherwise. Carlton's work confirms the robust abilities of RTS. Starting with arbitrarily chosen initial solutions, his RTS consistently achieved feasible solutions within one percent of the optimal solution when applied to the Solomon data set. He then found feasible starting tours produced better overall solutions using less computational effort. (Carlton 1995)

Recent improvements to TS include the addition of compound moves to the neighborhood search and parallel tabu searches that share information. The results of Rego and Roucairol in their 1996 article suggest Carlton's RTS could be improved with these techniques. Glover's 1995 *Tabu Search Fundamentals and Uses* is a useful reference and also suggests the use of compound moves and parallel processing as avenues of improvement.

K.3. Embedded Optimization

Despite the wealth of real-world application, examples of embedded optimization in the literature are rare (Hall 1997). Kassou and Pecuchet (1994) apply embedded optimization to job shop scheduling, where their object-oriented programming application uses a sophisticated optimization framework with an extensive user interface. Using the

optimization routines within a simulation to provide possible scheduling scenarios, the authors arrive at "guided rules" for choosing one of the three optimization techniques available and how to guide the search. Kassou and Pecuchet (1994) introduce a feedback loop between the optimization search and the simulation processes, but the nature of the information shared is ambiguously defined and the user must maintain interface in the loop (even to the point of being the "Generator of rules").

Brown and Graves (1981) furnish an example that does not adhere to our definition of embedded optimization, when they use optimization routines to replace time-consuming manual operations for the routing decisions of a nation-wide fleet of petroleum tank trucks. Whereas Brown and Graves refer to their structure as "embedded optimization," their work better exemplifies an "application" of optimization routines where none were used previously, and not the embedding of optimization routines as an event within a simulation.

Most current software fails to move beyond the constraint of user-defined "what-if" situations (Hall and Partyka 1997). As a counter-example, Glover, Laguna, and Kelly (1996) provide a good example of embedded optimization in a simulation that calls upon Glover's scatter search (1977) and tabu search heuristics to find near-optimal solutions. A neural-net "accelerator" may be used to cull out input combinations that the neural net learns will generate poor solution quality.

K.4. MODSIM

CACI's MODSIM programming language is an object-oriented language that lends itself to this approach. Much more than a traditional data structure or subroutine, a

MODSIM object can contain its own fields and routines, called methods. Marti's text, not yet published, and CACI's MODSIM III Tutorial and User's Guide were helpful resources in the coding of the RTS objects.

K.5. Conclusions

Battiti's reactive tabu search and the version created by Carlton are powerful heuristics for the VRPTW. Given the many directions one can take in GVRP research, object-oriented programming is a necessary coding methodology. Stochastic versions of GVRP problems significantly increase the complexity and have been largely avoided. Embedded optimization poses a powerful remedy for this untapped area. Although a large body of research and software addresses the GVRP, considerable work remains, especially for military applications.

Appendix J: MuavEval

A second step to MuavThreat2, the main module MuavEval runs the *evaluation phase* of UAV problems with stochastic winds, service times, and threats.

```
MAIN MODULE uavEval;

FROM IOMod IMPORT StreamObj, ALL FileUseType, ReadKey;
FROM OSMod IMPORT SystemTime;
FROM MathMod IMPORT pi;

FROM uavMod IMPORT timeMatrixObj;
FROM twReduceMod IMPORT twReductionObj;
FROM uavMod IMPORT startUAVObj;
FROM uavMod IMPORT uavRTSobj;

FROM tabuMod IMPORT arrReal2dimType;
FROM tabuMod IMPORT coordArrType;
FROM tabuMod IMPORT tourType;
FROM tabuMod IMPORT nodeType;
FROM tabuMod IMPORT vrpPenType;
FROM tabuMod IMPORT arrInt2dimType;
FROM tabuMod IMPORT arrIntType;
FROM tabuMod IMPORT arrRealType;

FROM tabuMod IMPORT SwapNode;
FROM uavMod IMPORT twCvrgServToFile;
FROM tabuMod IMPORT LatLongToFile;
FROM tabuMod IMPORT qcktourFile;
FROM tabuMod IMPORT tourToScreen;
FROM tabuMod IMPORT timeToFile;
FROM tabuMod IMPORT tourSched;
FROM tabuMod IMPORT countVeh;

FROM uavMod IMPORT expCvrg;

FROM RandMod IMPORT RandomObj, SetSeed, FetchSeed;

VAR
        timeMatrix : timeMatrixObj;
        twReduce : twReductionObj;
        startTour : startUAVObj;
        rts : uavRTSobj;
        randObj1, randObj2, randObj3, randObj4, randObj5 : RandomObj;

        instrm,
        instrm2, instrm3, instrm4,
        outstrm,
        outstrm2,
        outInit : StreamObj;
```

```
factor,   {used to convert the time windows to integer values}
TWPEN,          {Penalty weight assigned to the sum of late arr TW violations}
INCREASE,       {RTS parameter: mult. factor to decrease tabu length}
DECREASE,       {RTS parameter: mult. factor to increase tabu length}

windconv,   {multiplied by the resulting UAV time matrix, it provides an
            integer matrix (for calc speed) with the needed precision}
sumTij,         {sum of the i to j distances in the distance matrix}
mindist, {minimum travel distance}
maxdist, {maximum travel distance}
distAvg, {avg travel distance}
wdir,           {direction of wind vector}
ploiter,  {probabilitiy you loiter over a target}
loiter,         {loiter? - individual node result}
riskadj,  {amount to randomly adjust a target's prob of survival}
PSFCT,          {factor multiplied by coverage results to get more info into
                the integer move value}
cvrg,           {expected coverage of the tour}
bfCvrg,  {exp coverages of best and best feas tours}
bestCvrg,
servSum         {sum of increase over minimum service times}

        : REAL;

i, j, k,
endnum,         {end number in a numbered data file group}
maxtime,        {max possible time of arrival to any node, for time read}
numcycles,      {number of TW reduction cycles wanted}
numchanges,     {number of TWs reduced by TW reduction Obj}
numnodes,       {number of nodes in the directed graph}
nv,             {number of vehicles}
nc,             {number of targets/customers}
gamma,          {arbitrary cost assigned to the use of each vehicle}
iters,          {number of Tabu Search Iterations per problem}
tourLen, {Length of tour in time}
tvl,            {travel time of tour}
totPenalty,     {Total Penalty assigned to current tour}
tourCost,       {tour Length + Time Window Cost}
penTrav,        {tourCost - totWait == travel time + TW penalty}
bfCost,         {lowest tourCost found for a feasible tour}
bestCost,       {lowest tourCost found for a any tour}
bestTT,         {lowest travel time found for a any tour}
bestnv,         {# vehs used by best overall tour}
bfTT,           {lowest travel time found for feasible tour}
bfnv,           {# vehs used by best feas tour}
bfiter,   {iteration # when best feasible tour found}
tourhv,         {tour's hashing value}
bestiter, {iteration the best Tour found}
bestTime,       {Time the best Tour found}
bfTime,         {Time the best feasible Tour found}
numfeas,        {number of feasible solns found}
startTime,      {start Time (after time matrix, before TW reductions)}
stopTime,       {stop Time (after last iteration)}

DEPTH,          {depth of nodes we look for insert moves}
```

```
ZRANGE,             {upper bound on random integer weights assigned to nodes}
HTSIZE,             {size of hash table array}
CYMAX,              {max cyleLength used to alter mavg}
tabuLen, {current length of tabu tenure}
minTL,          {minimum Tabu Length}
maxTL,          {maximum Tabu Length}

wmag,           {magnitude of wind vector}
as,             {UAV's air speed}
numdays,        {number of days to run random scenarios}
day,            {index of current day}
nInitdays,      {number of days in the initialization set}
totaldays,      {nInitdays + numdays}
nvInit,         {# vehicles in initial tour read from a file}
nvu,            {# vehicles used in current tour}

windques,       {ask whether or not you want random winds}
magseed, dirseed,           {seeds for random winds}
startques,      {ask whether or not you want to input the initial tour}
servseed, loitseed,         {seeds for random service times}
servques,       {ask whether or not you want random service times}
initques, {ask if an initialization set already performed}
riskques,       {ask if random service times are needed}
cvrseed,

lowdeg,         {low end of range of wind direction to test}
highdeg, {high end of range of wind direction to test}
lowmag,         {low end of range of wind magnitude to test}
highmag,        {high end of range of wind magnitude to test}
minloiter,
maxloiter,      {minimum & maximum loiter time}

dayscore,       {robustness score of day under consideration}
maxdayscore,    {max robustness measure found}
bestscore,      {robustness measure of best route found}
sumScores,      {sum of all dayscores, used to find a mean}
robustChoice,   {tags the resulting tour chosen as most robust}

startday, {# of day beginning the scenario, day, loop}
rday      {loop var of the incremental robust tour choice}
          : INTEGER;

outfile,  {name of output file}
where,          {where in the code?}
str,

startfile,
file, filein,   {filenames}
filebegin,
fileout3,
fileout2,
filetour, filefreq,
fileout : STRING;

loadprint,          {print load on vehicles}
```

```
        stepprint,              {print each move evaluation}
        moveprint,              {print every insert move made by RTS}
        startprint,             {print starting tour and tw reduction steps}
        cycleprint,             {print hash results}
        timeprint,              {print time matrix}
        twrdprint : BOOLEAN;    {print tw reduction steps}

        psurv    : arrRealType;          {prob of survival array}

        coord    : coordArrType;                 {coordinates array}

        bfTour,                                  {best feasible tour found}
        bestTour,                                {node array holding best tour}
        tour,                                    {node array holding the tour}
        oldtour,                         {temporary tour}
        inittour  : tourType;                    {tour to read in an initial tour}

        tourPen   : vrpPenType;          {record of curr tour penalties}

        windmag,                         {array of wind magnitude per day}
        winddir,                {array of wind direction per day}
        duration,               {array of time to best solution per day}
        besttype,               {array tracking type of best: 1=feas, 0=not}
        scores,                          {array of robustness scores}

        m,                               {array of TW midpoints}
        slo, shi,               {arrays of service time ranges}
        s        : arrIntType;  {service times used}

        dist     : arrReal2dimType;              {no wind distance matrix}

        temp,

        routefreq,                       {counts the frequency that route i to j
                                          is chosen, where i and j are the array
                                          indices, in that order}
        time     : arrInt2dimType;               {time matrix}

        tourChoice : ARRAY INTEGER OF tourType; {array of tour choices per day}
        psurvDay : ARRAY INTEGER OF arrRealType; {array of psurv per day}

        node : nodeType;


BEGIN

        {INITIALIZE}
        startprint := FALSE;    {print starting tour}
        timeprint := FALSE;     {print time matrix}
        stepprint := FALSE;     {print each RTS step eval}
        moveprint := FALSE;     {print each RTS insert move}
        twrdprint := FALSE;     {print TW reduction steps}
        cycleprint := FALSE;    {print cycle/nocycle steps}
        loadprint := FALSE;     {print quantity & vehicle loads}
```

```
OUTPUT(" ");
OUTPUT("Please input the problem to work on:");
INPUT(file);

        NEW(instrm);                                    {open problem file}
        NEW(outstrm);                                   {open results file}
        NEW(outInit);                      {open file for future initial tour}
        filein := file + ".DAT";
        fileout := file + ".OUT";
        fileout2 := file + "Init.OUT";
        ASK instrm Open(filein, Input);
        ASK outstrm Open(fileout, Output);
        ASK outInit Open(fileout2, Output);

        fileout3 := file + "Rslt" + ".OUT";
        NEW(outstrm2);
        ASK outstrm2 Open(fileout3, Output);

str := "FILE: " + file;
ASK outstrm WriteString(str); ASK outstrm WriteLn; ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input the factor (such as 1, 10, 100, etc.) necessary to convert");
OUTPUT("the time window info to integer quantities");
INPUT(factor);

str := "Factor used for target windows and distances " + REALTOSTR(factor);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input the number of vehicles");
INPUT(nv);

        NEW(timeMatrix);


OUTPUT(" ");
OUTPUT("Do you want random service times?");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(servques);

IF servques = 1

  OUTPUT(" ");
  OUTPUT("Input seed number to use for service time randomization");
  INPUT(servseed);
  OUTPUT(" ");
  OUTPUT("Input seed number to use for loiter randomization");
  INPUT(loitseed);

  NEW(randObj3); NEW(randObj4);
  ASK randObj3 SetSeed(FetchSeed(loitseed));
  ASK randObj4 SetSeed(FetchSeed(servseed));
```

```
  OUTPUT(" ");
  OUTPUT("Give the probability you will loiter over a target");
  INPUT(ploiter);

 ASK outstrm WriteLn;
 str := "loitseed="+INTTOSTR(loitseed)+" servseed="+INTTOSTR(servseed)+
      " Pr{loiter} = "+REALTOSTR(ploiter);
 ASK outstrm WriteString(str); ASK outstrm WriteLn;

        {Reads in a coords in miles scenario with Service time ranges and psurv}
        ASK timeMatrix readUAVloiter(instrm, nc, numnodes, factor, nv,
                                     psurv, coord, tour, slo, shi,
                                     outstrm, startprint);

        NEW(s, 0..nc);
        s[0] := 0;

 ELSE

        {reads UAV file, finds nc, inits coord & tour}
        ASK timeMatrix readUAV(instrm, nc, numnodes, factor, nv,
                               psurv, coord, tour, s, outstrm,
                                  startprint);


 END IF;

        ASK instrm Close;        DISPOSE(instrm);

        {compute distance matrix, given coordinates in miles}
        ASK timeMatrix distMatrix(nc, numnodes, coord, dist, outstrm);

 IF timeprint
 {output distance matrix}
 NEW(temp, 0..numnodes, 0..numnodes);
 where := "No wind distance Matrix complete";
 FOR i := 0 TO numnodes
  FOR j := i+1 TO numnodes
    temp[i][j] := TRUNC(dist[i][j]);
    temp[j][i] := temp[i][j];
  END FOR;
 END FOR;
 timeToFile(where, outstrm, temp, numnodes);
 DISPOSE(temp);
 END IF;

        mindist := 9999.0; maxdist := 0.0;
        sumTij := 0.0; distAvg := 0.0;
        FOR i := 0 TO nc
          FOR j := i+1 TO nc
            sumTij := sumTij + dist[i][j];
            IF (dist[i][j] < mindist) AND (dist[i][j] > 0.0)
                        mindist := dist[i][j]; END IF;
            IF dist[i][j] > maxdist
                        maxdist := dist[i][j]; END IF;
```

```
        END FOR;
      END FOR;

      distAvg := sumTij / (FLOAT((nc+1)*(nc+1))/2.0 - FLOAT(nc+1) );

      OUTPUT(" ");
      OUTPUT("Average distance to travel is ", distAvg);
      OUTPUT("Min distance to travel is ", mindist);
      OUTPUT("Max distance to travel is ", maxdist);
      str := "Average distance to travel is " + REALTOSTR(distAvg);
      ASK outstrm WriteString(str); ASK outstrm WriteLn;
      str := "Min distance to travel is " + REALTOSTR(mindist);
      ASK outstrm WriteString(str); ASK outstrm WriteLn;
      str := "Max distance to travel is " + REALTOSTR(maxdist);
      ASK outstrm WriteString(str); ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input vehicle's air speed (in mi/hr)");
INPUT(as);

OUTPUT(" ");
OUTPUT("Please input the conversion factor to use with the WIND time matrix");
OUTPUT("The time windows will be updated to ensure the conversion matches");
OUTPUT("  (must be at least as great as previous factor)");
INPUT(windconv);

      {Update tour with windconv to match times}
      FOR i := 0 TO numnodes
            IF i <= nc
                  slo[i] := TRUNC(windconv / factor * FLOAT(slo[i]));
                  shi[i] := TRUNC(windconv / factor * FLOAT(shi[i]));
            END IF;

            tour[i].ea := TRUNC(windconv / factor * FLOAT(tour[i].ea));
            tour[i].la := TRUNC(windconv / factor * FLOAT(tour[i].la));

            IF tour[i].type = 2
              tour[i].arr := tour[i].ea;
              tour[i].dep := tour[i].arr;
            END IF;

      END FOR;

str := "Air speed " + REALTOSTR(as);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
str := "Factor used to make the wind time matrix integer" + REALTOSTR(windconv);
ASK outstrm WriteString(str); ASK outstrm WriteLn;
ASK outstrm WriteLn;


OUTPUT(" ");
OUTPUT("Please input the number of tabu search iterations");
OUTPUT("you would like to step through.");
INPUT(iters);
```

```
ASK outstrm WriteLn;
str :="# Iters = " + INTTOSTR(iters);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

OUTPUT(" ");
OUTPUT("Please input the number of days for which you would like to ");
OUTPUT("test random scenarios.");
INPUT(numdays);

OUTPUT(" ");
OUTPUT("Do you want random wind effects on every day");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(windques);

IF windques = 1

  OUTPUT(" ");
  OUTPUT("Input seed number to use for wind mag");
  INPUT(magseed);
  OUTPUT(" ");
  OUTPUT("Input seed number to use for wind dir");
  INPUT(dirseed);

ASK outstrm WriteLn;
str := "magseed="+INTTOSTR(magseed)+" dirseed="+INTTOSTR(dirseed);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

  NEW(randObj1);  ASK randObj1 SetSeed(FetchSeed(magseed));
  NEW(randObj2);  ASK randObj2 SetSeed(FetchSeed(dirseed));

  OUTPUT(" ");
  OUTPUT("Please input the range of DEGREES you would like to test");
  OUTPUT("  - Put lowest number first");
  OUTPUT("  - If testing winds around the 0 deg direction,");
  OUTPUT("    Make sure lowdeg is negative");
  INPUT(lowdeg);
  INPUT(highdeg);

  OUTPUT(" ");
  OUTPUT("Please input the range of MAGNITUDE you would like to test");
  OUTPUT("  - Put lowest number first");
  INPUT(lowmag);
  INPUT(highmag);

ASK outstrm WriteLn;
str :="RANDOM WINDS: degrees " + INTTOSTR(lowdeg) + " " + INTTOSTR(highdeg)
        + " magnitude " + INTTOSTR(lowmag) + " " + INTTOSTR(highmag);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

ELSE

  OUTPUT(" ");
  OUTPUT("Please input the magnitude of the wind vector (in mi/hr)");
  INPUT(wmag);
```

```
  OUTPUT(" ");
  OUTPUT("Please input the direction that the wind is blowing FROM in degrees");
  OUTPUT("  (due EAST is 0 degs, due NORTH is 90 degs, and so on)");
  INPUT(wdir);

  wdir := pi / 180.0 * wdir;

END IF;

OUTPUT(" ");
OUTPUT("Do you want to input the initial tour?");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(startques);

IF startques = 1

  OUTPUT(" ");
  OUTPUT("Input the file from which to read the initial tour");
  INPUT(startfile);

  {open problem file}
  NEW(instrm2);

  filein := startfile + ".DAT";
  ASK instrm2 Open(filein, Input);

  {initialize array of node id's}
  NEW(m, 0..numnodes);
  FOR j := 1 TO nc
        m[j] := 0;
  END FOR;

  ASK instrm2 ReadInt(nvInit);

  IF nvInit <> nv
    OUTPUT("nv and # vehicles in initial tour do not agree -- Break program!!");
  END IF;

  FOR i := 0 TO numnodes
    ASK instrm2 ReadInt(m[i]);  {m contains the id at position i}
  END FOR;

  ASK instrm2 Close;        DISPOSE(instrm2);

  OUTPUT(" ");
  OUTPUT("Do you want to input an initialization set?");
  OUTPUT("  - 1 = YES");
  OUTPUT("  - 0 = NO");
  INPUT(initques);


  IF initques = 0
   totaldays := numdays;
```

```
 nInitdays := 0;
 startday := 1;

ELSE

 OUTPUT(" ");
 OUTPUT("Input the name of the file of tour arrays");
 INPUT(filetour);

 OUTPUT(" ");
 OUTPUT("Input the name of the file of the route frequency matrix");
 INPUT(filefreq);

 NEW(instrm3); NEW(instrm4);

 filetour := filetour + ".DAT";
 ASK instrm3 Open(filetour, Input);

 filefreq := filefreq + ".DAT";
 ASK instrm4 Open(filefreq, Input);

 {Read in the number of tours in the initialization set}
 ASK instrm3 ReadInt(nInitdays);
 totaldays := nInitdays + numdays;
 startday := nInitdays + 1;

 NEW(tourChoice, 1..totaldays, 0..numnodes);

 {Read in the initialization set}
 FOR i := 1 TO nInitdays
  NEW(oldtour, 0..numnodes);

      FOR j := 0 TO numnodes

        NEW(node);
        oldtour[j] := node;

        ASK instrm3 ReadInt(oldtour[j].id);
        {set node types}
        IF (oldtour[j].id = 0) OR (oldtour[j].id > nc)
              oldtour[j].type := 2;          {2=veh node}
        ELSE
          oldtour[j].type := 1;     {1=cust node}
        END IF;

        oldtour[j].ea := tour[oldtour[j].id].ea;
        oldtour[j].la := tour[oldtour[j].id].la;
        oldtour[j].dep := tour[oldtour[j].id].dep;
        oldtour[j].arr := tour[oldtour[j].id].arr;

        tourChoice[i][j] := CLONE(oldtour[j]);

      END FOR;
 END FOR;
```

```
OUTPUT("aa");

   {Read in the route frequency matrix of the initialization set}
   NEW(routefreq, 0..numnodes, 0..numnodes);
   FOR i := 0 TO numnodes
         FOR j := 0 TO numnodes
           ASK instrm4 ReadInt(routefreq[i][j]);
         END FOR;
   END FOR;

   {output route frequency matrix}
   where := "ROUTE FREQUENCY MATRIX, History included ";
   timeToFile(where, outstrm, routefreq, numnodes);

   ASK instrm3 Close; ASK instrm4 Close;
   DISPOSE(instrm3); DISPOSE(instrm4);

 END IF;  {initques}

END IF;


OUTPUT(" ");
OUTPUT("Do you want random THREAT effects on every day");
OUTPUT("  - 1 = YES");
OUTPUT("  - 0 = NO");
INPUT(riskques);

OUTPUT(" ");
OUTPUT("Input the factor to convert coverage to an integer value");
INPUT(PSFCT);

IF riskques = 1

 OUTPUT(" ");
 OUTPUT("Input seed number to use for random COVERAGES");
 INPUT(cvrseed);

ASK outstrm WriteLn;
str := "RANDOM THREATS: cvrseed="+INTTOSTR(cvrseed);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

 NEW(randObj5);  ASK randObj5 SetSeed(FetchSeed(cvrseed));

END IF;


OUTPUT(file);
OUTPUT(filein);
OUTPUT(fileout);
OUTPUT(fileout2);

{*} {denotes a parameter setting}
{*       nv := 10;            *}
{*       windconv := 10.0; *}
```

```
{*      iters := 1000;   *}
{*}     TWPEN := 10.0;
{*}     gamma := 0;

{*}     INCREASE := 1.2;
{*}     DECREASE := 0.9;
{*}     CYMAX := 50;
{*}     HTSIZE := 131073;
{*}     ZRANGE := 1009;
{*}     minTL := 5;
{*}     maxTL := 2000;


{*}     DEPTH := nc+nv-1;
{*}     tabuLen := MINOF(30, nc+nv-1);

        {**** LOOP OF SCENARIOS ***}

        NEW(windmag, startday..totaldays);
        NEW(winddir, startday..totaldays);
        NEW(duration, startday..totaldays);
        NEW(psurvDay, startday..totaldays, 0..nc);
        NEW(scores, 1..totaldays);

        NEW(besttype, 1..totaldays);

        IF initques = 1
     FOR i := 1 TO nInitdays
           besttype[i] := 1;  {since only feasible tours used}
         END FOR;
        END IF;

        IF initques = 0
         NEW(tourChoice, 1..numdays, 0..numnodes);
         NEW(routefreq, 0..numnodes, 0..numnodes);

         {initialize matrix of route frequency counts}
         FOR i := 0 TO numnodes
          FOR j := 0 TO numnodes
                routefreq[i][j] := 0;
          END FOR;
         END FOR;
        END IF;

        NEW(startTour); {find initial tour and/or initial penalties}


{***}   FOR day := startday TO totaldays           {**********}

ASK outstrm WriteLn;
str :="DAY: " + INTTOSTR(day);
ASK outstrm WriteString(str); ASK outstrm WriteLn;

          IF windques = 1
           windmag[day] := ASK randObj1 UniformInt(lowmag, highmag);
```

```
            wmag := windmag[day];
            winddir[day] := ASK randObj2 UniformInt(lowdeg, highdeg);
            wdir := FLOAT(winddir[day]);
            wdir := pi / 180.0 * wdir;
          END IF;

ASK outstrm WriteLn;
str :="WIND: magnitude ="+INTTOSTR(wmag)+" direction(rads) = " + REALTOSTR(wdir);
ASK outstrm WriteString(str); ASK outstrm WriteLn; ASK outstrm WriteLn;


          IF riskques = 1

            {randomly adjust the prob of survival at the target nodes}
            FOR i := 1 TO nc
              riskadj := ASK randObj3 UniformReal(-1.0, 1.0);
                  IF riskadj < -0.333
                          riskadj := -1.0;
                  ELSIF riskadj > 0.333
                          riskadj := 1.0;
         ELSE
                          riskadj := 0.0;
                  END IF;
              psurvDay[day][i] := psurv[i] + riskadj / 10.0;

            END FOR;

          ELSE
                  psurvDay[day] := psurv;
          END IF;
OUTPUT("aaa");

          IF servques = 1

            FOR i := 1 TO nc

              loiter := ASK randObj3 UniformReal(0.0, 1.0);

              IF loiter <= ploiter
                s[i] := ASK randObj4 UniformInt(slo[i], shi[i]);
              ELSE
                s[i] := slo[i];
              END IF;

            END FOR;

          END IF;

          ASK timeMatrix timeMatrix(nc, numnodes, gamma, as, wmag, wdir, windconv,
                                    coord, s, dist, time, outstrm, timeprint);
OUTPUT("b");

          IF day = startday {must find a true initial tour}

            IF startques = 1          {read in initial tour}
```

```
            NEW(inittour, 0..numnodes);

            {Reorder tour, currently in numerical order, by the initial tour
             and place temporarily into inittour}

            FOR i := 0 TO numnodes
             inittour[i] := CLONE(tour[m[i]]);
            END FOR;

            {Copy inittour into tour}
            FOR i := 0 TO numnodes
             tour[i] := CLONE(inittour[i]);
            END FOR;

            DISPOSE(inittour);

            tourSched(1, nc, numnodes, tour, time, tourLen, outstrm);

            startTime := SystemTime();

      ELSE

        ASK startTour startTour(nv, nc, time, tour, tourLen,
                                      totPenalty, tourhv, startTime, m, outstrm);
      END IF;

      DISPOSE(m);

    ELSE

      IF initques = 0

        {lose old tour, use previous days Choice}
        DISPOSE(tour);
        NEW(tour, 0.. numnodes);
        {use the best result of the previous day}
   FOR i := 0 TO numnodes
    tour[i] := CLONE(tourChoice[day-1][i]);
   END FOR;

      ELSE
OUTPUT("c");
            {use the robust tour, But it may not have TW info with it}
            NEW(inittour, 0..numnodes);

            {So reorder the tour of the previous day into inittour from the robust tour}

            FOR i := 0 TO numnodes
             inittour[i] := CLONE(tour[tourChoice[robustChoice][i].id]);
            END FOR;

            {Copy inittour into tour}
            FOR i := 0 TO numnodes
             tour[i] := CLONE(inittour[i]);
```

```
                END FOR;

                DISPOSE(inittour);
            END IF;

            {Compute initial schedule, return tour's total travel + wait time}
            tourSched(1, nc, numnodes, tour, time, tourLen, outstrm);

            startTime := SystemTime();

        END IF;
OUTPUT("d");
IF startprint
{* ASK outstrm WriteString("startTour complete"); ASK outstrm WriteLn;
qcktourFile(outstrm, tour, numnodes); *}
where := "startTour complete";
twCvrgServToFile(where, outstrm, tour, nc, numnodes, tourLen,
            windconv, loadprint, psurv, s, slo);
END IF;
          ASK startTour startUAVbest(numnodes, tvl, tourLen, tour, TWPEN,
                                     psurvDay[day], totPenalty, penTrav, tourCost,
                                     tourPen, bfiter, bfCost, bfTT, bfnv, bestiter,
                                     bestCost, bestTT, bestnv, bfTime,
                                     bestTime, cvrg, bfCvrg, bestCvrg,
                                     bestTour, bfTour);
OUTPUT("e");
          NEW(rts);
          {conduct RTS}
          ASK rts search(psurvDay[day], PSFCT,
                         TWPEN, INCREASE, DECREASE, HTSIZE, CYMAX, ZRANGE, DEPTH,
                         minTL, maxTL, tabuLen, iters, nc, numnodes,
                         outstrm, outstrm2, tourPen, time, stepprint,
                         moveprint, cycleprint, tourCost, penTrav, totPenalty, tvl,
                         bfCost, bfTT, bfnv, bfiter, bestCost, bestTT, bestnv,
                         bestTime, bfTime, bestiter, numfeas,
                         bfCvrg, bestCvrg, cvrg,
                         tour, bestTour, bfTour);
          DISPOSE(rts);

          stopTime := SystemTime();

          IF bfiter > -1

           {save the best feasible tour found}
           FOR i := 0 TO numnodes
             tourChoice[day][i] := CLONE(bfTour[i]);
           END FOR;

           {output the results}
           where := "DAY " + INTTOSTR(day) + "  BEST FEASIBLE TOUR";
           twCvrgServToFile(where, outstrm, bfTour, nc, numnodes, bfCost,
                                factor, loadprint, psurvDay[day], s, slo);

           duration[day] := bfTime - startTime;
           besttype[day] := 1;
```

```
            ASK outstrm WriteString("# vehicles used = ");
            ASK outstrm WriteInt(bfnv, 2); ASK outstrm WriteLn;

            ASK outstrm WriteString("Best Feasible solution found after ");
            ASK outstrm WriteString(INTTOSTR(bfTime-startTime)+" secs");
            ASK outstrm WriteLn;
            ASK outstrm WriteString("on Iteration: "+ INTTOSTR(bfiter));
            ASK outstrm WriteLn;
            ASK outstrm WriteString("with travel time = "+ INTTOSTR(bfTT));
            ASK outstrm WriteLn;
            ASK outstrm WriteLn;
            ASK outstrm WriteString("& Expected coverage = "+REALTOSTR(bfCvrg));
            ASK outstrm WriteLn;

            {update the route frequency matrix}
            FOR i := 0 TO numnodes-1
                  j := bfTour[i].id;
                  k := bfTour[i+1].id;
                  routefreq[j][k] := routefreq[j][k] + 1;
            END FOR;

OUTPUT("f");
          ELSE

            {save the best tour found}
            FOR i := 0 TO numnodes
              tourChoice[day][i] := CLONE(bestTour[i]);
            END FOR;

            {output the results}
            where := "DAY " + INTTOSTR(day)
                          + "  Search complete: BEST TOUR (NOT FEASIBLE)";
            twCvrgServToFile(where, outstrm, bestTour, nc, numnodes, bestCost,
                    windconv, loadprint, psurvDay[day], s, slo);

            duration[day] := bestTime - startTime;
            besttype[day] := 0;

            ASK outstrm WriteString("# vehicles used = ");
            ASK outstrm WriteInt(bestnv, 2); ASK outstrm WriteLn;
            ASK outstrm WriteString("Best solution found after ");
            ASK outstrm WriteString(INTTOSTR(bestTime-startTime)+" secs");
            ASK outstrm WriteLn;
            ASK outstrm WriteString("on Iteration: "+ INTTOSTR(bestiter));
            ASK outstrm WriteLn;
            ASK outstrm WriteString("with travel time = "+ INTTOSTR(bestTT));
            ASK outstrm WriteLn;
            ASK outstrm WriteLn;
            ASK outstrm WriteString("& Expected coverage = "+REALTOSTR(bestCvrg));
            ASK outstrm WriteLn;
{** DONT UPDATE FREQ MATRIX WITH BAD TOUR
            {update the route frequency matrix}
            FOR i := 0 TO numnodes-1
                  j := bestTour[i].id;
```

```
            k := bestTour[i+1].id;
            routefreq[j][k] := routefreq[j][k] + 1;
        END FOR;
****}
       END IF;

       {Output service time difference}
       servSum := 0.0;
       FOR i := 1 TO nc
         servSum := servSum + FLOAT(s[i] - slo[i]);
       END FOR;
       servSum := servSum / windconv;
       str := "Sum of increase over min service times = "+ REALTOSTR(servSum);
       ASK outstrm WriteLn; ASK outstrm WriteString(str);
       ASK outstrm WriteLn;

       {If we're in the test set, find the Robust Tour every day}
       IF initques = 1
             {find most robust tour chosen}
             robustChoice := 1;
             dayscore := 0;
             maxdayscore := 0;
             sumScores := 0;

             FOR rday := 1 TO day

               FOR i := 0 TO numnodes-1

                 dayscore := dayscore
             + routefreq[tourChoice[rday][i].id][tourChoice[rday][i+1].id];
               END FOR;
               scores[rday] := dayscore;
               sumScores := sumScores + dayscore;

               {choose the tour with the most robust routes}
               IF dayscore > maxdayscore
                 robustChoice := rday;
                 bestscore := dayscore;
                 maxdayscore := dayscore;

               ELSIF dayscore = maxdayscore

                 {choose feasible tours over nonfeas, or choose the most recent}
                 IF besttype[rday] >= besttype[robustChoice]
                   robustChoice := rday;
                   bestscore := dayscore;
                 END IF;

               END IF;

               dayscore := 0;

             END FOR; {rday}

             str := "Day = "+INTTOSTR(day)+" and Robust Choice = "
```

```
                +INTTOSTR(robustChoice);
          ASK outstrm WriteLn; ASK outstrm WriteString(str);
          ASK outstrm WriteLn;

      END IF;

      {output coords to file so we can scatter plot tours}
      where := "Day = " + INTTOSTR(day);
      LatLongToFile(where, outstrm2, tourChoice[day], nc, numnodes, coord);

      DISPOSE(bfTour);
      DISPOSE(bestTour);
      DISPOSE(tourPen);

    END FOR;
    {**** END OF DAY LOOP ****}
OUTPUT("k");

    {output route frequency matrix}
    where := "SCENARIO LOOP COMPLETE, Frequency of Routes Chosen: ";
    timeToFile(where, outstrm, routefreq, numnodes);

    {find most robust tour chosen}
    dayscore := 0;
    maxdayscore := 0;
    sumScores := 0;
    robustChoice := 1;

    FOR day := 1 TO totaldays

      FOR i := 0 TO numnodes-1
        dayscore := dayscore
              + routefreq[tourChoice[day][i].id][tourChoice[day][i+1].id];
      END FOR;
      scores[day] := dayscore;
      sumScores := sumScores + dayscore;

      {choose the tour with the most robust routes}
      IF dayscore > maxdayscore
        robustChoice := day;
        bestscore := dayscore;
        maxdayscore := dayscore;

      ELSIF dayscore = maxdayscore

        {choose feasible tours over nonfeas, or choose the most recent}
        IF besttype[day] >= besttype[robustChoice]
          robustChoice := day;
          bestscore := dayscore;
        END IF;

      END IF;

      dayscore := 0;
```

```
	END FOR;  {DAY LOOP}

OUTPUT("1");
	{output robust tour}
	tourSched(1, nc, numnodes, tourChoice[robustChoice], time, tourLen, outstrm);
	countVeh(numnodes, tourChoice[robustChoice], nvu);
	expCvrg(numnodes, psurv, tourChoice[robustChoice], cvrg);

	where := "MOST ROBUST TOUR: day = " + INTTOSTR(robustChoice);
	twCvrgServToFile(where, outstrm, tourChoice[robustChoice], nc, numnodes,
		tourLen, windconv, loadprint, psurvDay[totaldays], s, slo);

	ASK outstrm WriteString("# vehicles used = ");
	ASK outstrm WriteInt(nvu, 2); ASK outstrm WriteLn;
	ASK outstrm WriteString("With robustness score "+ INTTOSTR(bestscore));
	ASK outstrm WriteLn; ASK outstrm WriteLn;
OUTPUT("2");
	ASK outstrm WriteString("MEAN robustness score "
			+ INTTOSTR(sumScores DIV totaldays));
	ASK outstrm WriteLn; ASK outstrm WriteLn;

	{Output Robustness scores}
	ASK outstrm WriteLn;
	ASK outstrm WriteString("Robustness scores: ");
	FOR i := 1 TO totaldays
	  ASK outstrm WriteInt(i, 3);
	  ASK outstrm WriteInt(scores[i], 5);
	  ASK outstrm WriteLn;
	END FOR;

	{Output Robust tour for future Initial tour}
	ASK outInit WriteInt(nv, 3); ASK outInit WriteLn;
	FOR i:= 0 TO numnodes
	  ASK outInit WriteInt(tourChoice[robustChoice][i].id, 5);
	  ASK outInit WriteLn;
	END FOR;

	ASK outstrm Close;
	ASK outInit Close;
    ASK outstrm2 Close;

	DISPOSE(startTour);
	DISPOSE(timeMatrix);
	DISPOSE(outstrm);
	DISPOSE(outInit);
	DISPOSE(outstrm2);
	DISPOSE(s);
	DISPOSE(time);
	DISPOSE(coord);

	IF windques = 1
	DISPOSE(randObj1); DISPOSE(randObj2);
	END IF;
	IF servques = 1
	DISPOSE(randObj3); DISPOSE(randObj4);
```

```
        END IF;

END MODULE; {MAIN}
```

Appendix K: Literature Review

This review concentrates on contemporary works relevant to the class of general vehicle routing problems (GVRP) approached within this thesis and works related to our embedded optimization approach. It is separated into three sections, although some works contribute to more than section. Section K.1 provides a summary of the literature relevant to an understanding of the GVRP and the attacked subset. Section K.2 follows with a survey of works describing tabu search (TS) and some relevant applications of this powerful meta-heuristic. Section K.3 reviews literature exploring the utility and relationships important to the embedded optimization of the TS heuristic within a simulation or other problem solving form. Section K.4 describes the resources used by the author as references for CACI's MODSIM programming lanquage. A number of MODSIM objects created by the researcher form a fundamental part of this study. Section K.5 closes with a few conclusions drawn from the literature review.

K.1. The General Vehicle Routing Problem

In their 1997 survey article of commercial vehicle routing software, Hall and Partyka describe the vehicle routing problem (VRP) as an interdependent collection of four problems. One is the calculation of distances between stops, two is the partitioning of the region into districts of feasible routes, three is usually a traveling salesman problem (TSP) or a TSP with time window constraints (TSPTW), and four is the subsequent crew assignment (Hall and Partyka 1997). Within the collection of articles entitled *Vehicle Routing: Method and Studies* (Golden and Assad 1988), Assad places the general characteristics of routing problems into six categories: nature of demand, information on

demand, vehicle fleet, crew requirements, scheduling requirements, and data requirements. Both descriptions illustrate the need for any routing system to incorporate elements of embedded optimization. While these descriptions capture the industrial application of the VRP, they fall short for most military applications as they do not include a variance in the nature of the operational environment.

In his thesis on unmanned aerial vehicle routing, Sisson created a formulation of a multiple vehicle TSPTW (mTSPTW) that incorporated the probabilities of vehicle attrition due to hostile forces into the objective function (Sisson 1997). No civilian approach considers the possibility of attrition due to a hostile environment. Although the commercial market is competitive, combative behavior remains illegal. Sisson also researched the performance of unmanned aerial vehicles the important influence of wind. Sisson's work is a critical stepping stone for this thesis.

As Hall and Partyka noted, an mTSPTW often comprises a critical portion of any larger VRP. To better understand the relationship of problems within the GVRP family, a number of works were consulted. *The Traveling Salesman Problem* holds a prominent standing amongst the literature available. In Chapter 2 of that collection, R. S. Garfinkel quickly summarizes the TSP's "seductive" qualities and provides five distinctly separate applications. Garfinkel also provides some simple transformations to the TSP, including the introduction of multiple salesman (referred to previously as multiple vehicle) and the relaxation allowing the repetition of cities. Garfinkel continues through the generalization of the TSP as an assignment problem and a minimum spanning tree, as well as providing a standard linear programming formulation. Chapter 12 of *The Traveling Salesman Problem*, written by N. Christofides, is entitled "Vehicle routing."

Christofides provides three formulations, some optimization algorithms, and some heuristics. Nemhauser and Wolsey's text, *Integer and Combinatorial Optimization*, provided an example meant for students of the subject after providing a integer programming formulation of the issues involved.

A more current survey of approaches to the TSP and VRP was accomplished in 1992 by Gilbert Laporte. In the first of two invited reviews for the *European Journal of Operational Research*, Laporte surveys the main exact and heuristic algorithms for the TSP. The second article performs the same function for the VRP. After introducing tabu search, Laporte states the "computational results indicate that the proposed heuristic may be one of the best ever developed for the VRP" (Laporte 1992b).

Although the TSP is classified as NP-hard (Glover 1997; Lenstra 1985), special cases exist that can be solved in polynomial time (Glover 1997). In their 1997 article, Glover and Punnen identify new solvable cases of the TSP. A similar work accomplished by the Optimization Group at the Technical University of Graz surveys known efficiently solved cases. This thesis suggests such cases be sought by any VRP software before a more general algorithm, such as tabu search, is applied.

Finally, Carlton's 1995 dissertation is a critical resource. Carlton surveys the proposed classification schemes of the problems within the GVRP class. He concludes that no prior system gives "a direct approach to GVRP classification to enhance the understanding and exploitation of the relationships among the GVRP problem types" (Carlton 1995). He then proposes a hierarchical taxonomy that classifies GVRP types along those lines. In brief, Carlton first summarizes the GVRP into three "floors." Each

floor includes the following cases and their possible combinations (Carlton's notation is slightly altered for more simplicity without loss of information):

1. SV: Single vehicle.
2. MVH: Multiple homogenous vehicles.
3. $\overline{\text{MVH}}$: Multiple non-homogenous vehicles
4. SD: Single depot.
5. MD: Multiple depots.
6. TW: Time window constraints present.
7. RL: Route length constraints present.

The first floor is the family of TSP problems. With the addition of vehicle capacity constraints, one transitions to the second floor of VRP problems. Precedence constraints cause a transition to third floor of pickup and delivery problems (PDP). As a tangible illustration of his success, this research employs Carlton's taxonomy to enact the object-oriented concepts of inheritance and polymorphism between MODSIM optimization objects.

The MODSIM objects accompanying this thesis correspond to the TSP, the mTSP, the mTSPTW, and the VRPTW. Carlton concentrates much his efforts upon the TSPTW and mTSPTW. Of course, a simple transformation creates TSPTW from the mTSPTW. From his literature review, Carlton concludes the TSPTW is not as "well studied" as the TSP and VRP; "it stands in the gap" (Carlton 1995). Carlton's focus on the TSPTW appears well warranted given the previous synopsis of "real-world" applications of the VRP given by Hall and Partyka. This research uses an adaptation of

Carlton's C programming language code for the VRPTW as a first step in the creation of the MODSIM objects.

Jaillet and Odoni (1988) demonstrate the added complexity of a probabilistic TSP (PTSP) over the TSP. For even a simple heuristic like the nearest-neighbor, they find the computational effort increases by $O(n^2)$ over the deterministic version. Furthermore, their formulation of the PTSP is simple in comparison to the UAV problem since they only consider the probability that customers are not present. Jaillet and Odoni sought the similar objective of "well-behaved" or robust routes, but all of their stochastic programming methods were bound to smaller numbers of customers by the necessary computational effort.

K.2. Tabu Search and Applications Related to the GVRP

As mentioned previously, Laporte's survey of VRP algorithms gave highest marks to tabu search (TS). He based this conclusion upon his own version of TS created with Gendreau and Hertz. In their 1996 article, Kervahut, Garcia, and Rousseau compare the performance of their TS heuristic to that of five other documented heuristics within the well known Solomon datasets. Their TS employed a tabu list of fixed length and infeasible regions were not accessible to the search. Yet, the quality of solutions reached by their version of TS bests all other considered heuristics except one known as GIDEON. A statistical test failed to reject the hypothesis that the tested TS heuristic and GIDEON are equally effective (Kervahut 1996). From the literature, it is apparent TS is a powerful heuristic within the GVRP class.

Given this justification for its use, an investigation of TS becomes imperative. Fred Glover introduced TS in 1986 and his writings form a necessary portion of any review regarding this heuristic. His 1990 article, "Tabu Search: A Tutorial," seemed an obvious place to start. Here, Glover provides guidelines in building a TS heuristic. Guidelines include suggestions for the length of tabu lists, the number of attributes considered, the comparison of attributes, the balancing of diversification and intensification efforts to the aspiration criteria, and the powers of target analysis.

As an application of artificial intelligence, target analysis is the application of empirical results from classes of problems to the problems attempted by your own heuristic. Glover strongly emphasizes to use of target analysis to improve move evaluations as his fifth guideline (Glover 1990a). His sixth guideline suggests the use of a frequency-derived (the frequency of revisited solutions) penalty to encourage diversification, with a possible marrying to restarting the search (Glover 1990a).

Glover had stated one year prior to the "tutorial" article in his "Tabu Search-Part 1" publication, that TS was "still in its infancy" (Glover 1989). The tutorial mostly provided a review of "where TS is, and where it is going" with Glover's guidelines, predictions, and other theoretical discussions for the meta-heuristic. Many of the guidelines were the obvious result of Glover's experience in TS application. Similarly, his predictions were educated guesses of things to come. Glover's "Tabu Search-Part I" and Tabu Search-Part II" are more heavily cited than the tutorial article, as they provide a more straight-forward and step-by-step presentation of TS (Glover 1989; 1990b).

With their reactive tabu search (RTS), Battiti and Tecchiolli provided the important next step suggested by Glover's fifth guideline. Battiti and Tecchiolli present

the reactive tabu search in their 1994 article and show it to be a far more robust procedure than the fixed and strict tabu search heuristics. To illustrate the technique's abilities, the authors apply it to the optimization of a quadratic assignment problem and a complex sinusoidal function. The applications are both successful and the authors are kind enough to provide detailed pseudocode. (Battiti 1994)

When one considers that the TS heuristics given by Laporte and Kervahut fall in the category of fixed TS and the RTS achieves significant jumps in computational efficiency without applying Glover's time consuming target analysis preprocessing, it beccomes apparent that Battiti and Tecchiolli have ushered in a powerful improvement to the application of TS. In his 1996 article, Battiti states that "parameter tuning" and "search confinement" are "potential drawbacks to simple implementations" of tabu search, genetic algorithms, and simulated annealing. He then explains that the reactive tabu search effectively overcomes these drawbacks. He classifies the reactive TS as a deterministic algorithm. However, it is quickly made stochastic through random tie breaking and a random "escape trajectory." (Battiti 1996)

Battiti shares the important find that the order of operations required for memory usage per iteration is O(1) or essentially a constant that is independent of the number of iterations performed. He also claims hashing techniques are available that need only a few bytes of memory and result in small "collision" probabilities. He does not support the claim, but he states small collision probabilities do not have statistically significant effects on the reactive TS heuristic. (Battiti 1996)

With a four 0/1 bit example, Battiti then illustrates the properties of reactive TS. In the example, the tabu list length takes progressively more iterations to

increase after the previous increment, and the distance of the search from the optimal "attractor" varies quickly and increases quickly to the maximum range. These illustrations are convincing evidence of the robust balance of intensification and diversification achieved with the reactive TS heuristic. (Battiti 1996)

Once again, Carlton's dissertation is helpful. He provides a two-level open hashing structure. This structure allows the TS to "efficiently store and accurately identify solutions in order to determine whether a particular solution has been visited previously during the search" (Carlton 1995). It minimizes the probability of two non-identical tours being incorrectly determined as identical (Carlton 1995).

It should be noted that Carlton's RTS is deterministic, while the RTS proposed by Battiti and Tecchiolli is stochastic. Carlton's RTS begins from a deterministic starting solution and uses deterministic escape routines, while the heuristic of Battiti and Tecchiolli begins from a stochastic starting solution and uses stochastic escape routines (Carlton 1995).

Although two and a half years have passed since the introduction of RTS, it remains at the forefront of TS heuristics proposed. A contrasting example is provided by Rochat and Taillard. In their 1995 article, the authors propose a technique to overcome the weaknesses of previous local searches and tabu search. The first weakness is the probability of becoming trapped in local optimum, the second is the large computational effort. The approach has two phases. The first begins with an initialization set generated from "good" heuristic solutions. The second phase seeks to extract good tours from the initial set (or from any previous tours after the 2nd iteration) and then seeks to improve this set. The improvement often arises from a combination of the previous "good" tours.

How these "good" solutions are achieved is not specified and appears to be a major weakness of the approach.

As it does with Glover's target analysis, RTS seems to obviate the need for any esoteric pre-processing of the sort used by Rochat and Taillard. These pre-processing techniques report impressive results, but they require a high computational cost and would be difficult to implement by non-TS experts working in the field vehicle routing, civilian and otherwise. Carlton's work confirms the robust abilities of RTS. Starting with arbitrarily chosen initial solutions, his RTS consistently achieved feasible solutions within one percent of the optimal solution when applied to the Solomon data set. He then found feasible starting tours produced better overall solutions using less computational effort. (Carlton 1995)

Recent improvements to TS include the addition of compound moves to the neighborhood search and parallel tabu searches that share information. The results of Rego and Roucairol in their 1996 article suggest Carlton's RTS could be improved with these techniques. Glover's 1995 *Tabu Search Fundamentals and Uses* is a useful reference and also suggests the use of compound moves and parallel processing as avenues of improvement.

K.3. Embedded Optimization

Despite the wealth of real-world application, examples of embedded optimization in the literature are rare (Hall 1997). Kassou and Pecuchet (1994) apply embedded optimization to job shop scheduling, where their object-oriented programming application uses a sophisticated optimization framework with an extensive user interface. Using the

optimization routines within a simulation to provide possible scheduling scenarios, the authors arrive at “guided rules” for choosing one of the three optimization techniques available and how to guide the search. Kassou and Pecuchet (1994) introduce a feedback loop between the optimization search and the simulation processes, but the nature of the information shared is ambiguously defined and the user must maintain interface in the loop (even to the point of being the “Generator of rules”).

Brown and Graves (1981) furnish an example that does not adhere to our definition of embedded optimization, when they use optimization routines to replace time-consuming manual operations for the routing decisions of a nation-wide fleet of petroleum tank trucks. Whereas Brown and Graves refer to their structure as “embedded optimization,” their work better exemplifies an “application” of optimization routines where none were used previously, and not the embedding of optimization routines as an event within a simulation.

Most current software fails to move beyond the constraint of user-defined “what-if” situations (Hall and Partyka 1997). As a counter-example, Glover, Laguna, and Kelly (1996) provide a good example of embedded optimization in a simulation that calls upon Glover’s scatter search (1977) and tabu search heuristics to find near-optimal solutions. A neural-net “accelerator” may be used to cull out input combinations that the neural net learns will generate poor solution quality.

K.4. MODSIM

CACI’s MODSIM programming language is an object-oriented language that lends itself to this approach. Much more than a traditional data structure or subroutine, a

MODSIM object can contain its own fields and routines, called methods. Marti's text, not yet published, and CACI's MODSIM III Tutorial and User's Guide were helpful resources in the coding of the RTS objects.

K.5. Conclusions

Battiti's reactive tabu search and the version created by Carlton are powerful heuristics for the VRPTW. Given the many directions one can take in GVRP research, object-oriented programming is a necessary coding methodology. Stochastic versions of GVRP problems significantly increase the complexity and have been largely avoided. Embedded optimization poses a powerful remedy for this untapped area. Although a large body of research and software addresses the GVRP, considerable work remains, especially for military applications.

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1998	Master's Thesis

4. TITLE AND SUBTITLE
Embedding a Reactive Tabu Search Heuristic
in Unmanned Aerial Vehicle Simulations

5. FUNDING NUMBERS

6. AUTHOR(S)
Capt. Joel L. Ryan, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AFIT/ENS
2950 P Street
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GOR/ENS/98M-21

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
USAF UAV BATTLELAB
ATTN: Maj Mark O'Hair
203 W. D Ave., Suite 406
Eglin AFB FL 32542-6867

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Advisor: Lt. Col. T. Glenn Bailey

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

We apply a Reactive Tabu Search (RTS) heuristic within a discrete-event simulation to solve routing problems for Unmanned Aerial Vehicles (UAVs). Our formulation represents this problem as a multiple Traveling Salesman Problem with time windows (mTSPTW), with the objective of attaining a specified level of target coverage using a minimum number of vehicles. Incorporating weather and probability of UAV survival at each target as random inputs, the RTS heuristic in the simulation searches for the best solution in each realization of the problem scenario in order to identify those routes that are robust to variations in weather, threat, or target service times.

Generalizing this approach as Embedded Optimization (EO), we define EO as a characteristic of a discrete-event simulation model that contains optimization or heuristic procedures that can affect the state of the system. The RTS algorithm in the UAV simulation demonstrates the utility of EO by determining the necessary fleet size for an operationally representative scenario. From our observation of robust routes, we suggest a methodology for using robust tours as initial solutions in subsequent replications. We present an object-oriented implementation of this approach using MODSIM III, and show how mapping object inheritance to the GVRP hierarchy allows for minimal adjustments from previously written objects when creating new types. Finally, we use EO to conduct an analysis of fleet size requirements within an operationally representative scenario

14. SUBJECT TERMS	15. NUMBER OF PAGES
Embedded Optimization; Tabu Search; Heuristics; Simulation; Optimization; Routing; Unmanned Aerial Vehicles	219
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL